
AdaptaMaterialEscolar: Herramienta para la adaptación
de asignaturas a necesidades educativas especiales

AdaptaMaterialEscolar: Tool for adapting subjects to
special educational needs



Trabajo de Fin de Grado

Pablo Miranda Torres
Natalia Rodríguez-Peral Valiente
Jorge Velasco Conde

Directoras

Virginia Francisco Gilmartín
Raquel Hervás Ballesteros

Trabajo de Fin de Grado en Ingeniería del Software
Facultad de Informática
Universidad Complutense de Madrid

Junio 2021

Documento maquetado con T_EX_S v.1.0+.

Este documento está preparado para ser imprimido a doble cara.

AdaptaMaterialEscolar: Herramienta para la adaptación de
asignaturas a necesidades educativas especiales
AdaptaMaterialEscolar: Tool for adapting subjects to
special educational needs

*Memoria que presentan para optar al título de Grado en Ingeniería
del Software*

Pablo Miranda Torres
Natalia Rodríguez-Peral Valiente
Jorge Velasco Conde

Dirigida por las Doctoras
Virginia Francisco Gilmartín
Raquel Hervás Ballesteros

Versión 1.0+

Trabajo de Fin de Grado en Ingeniería del Software
Facultad de Informática
Universidad Complutense de Madrid

Junio 2021

Agradecimientos

En primer lugar, queremos agradecer a nuestras directoras Virginia y Raquel por toda la dedicación y apoyo brindados en el proyecto, así como todos los consejos aportados para el desarrollo de AdaptaMaterialEscolar.

En segundo lugar queremos agradecer a nuestros familiares y amigos por todo el ánimo y ayuda que nos han proporcionado en esta etapa.

Por último, también queremos agradecer a todos los docentes del Aula TEA del Instituto Juan de Ávila por confiar en nosotros y darnos la oportunidad de crear de esta herramienta.

Resumen

Por ley, el currículum educativo trata de garantizar que el proceso de formación del alumnado cuente con los materiales, recursos y contenidos necesarios durante su recorrido académico, en igualdad de condiciones. Sin embargo, debido a que no todos los estudiantes cuentan con las mismas herramientas de aprendizaje, existe la posibilidad de realizar adaptaciones curriculares.

Actualmente no existen herramientas gratuitas para realizar adaptaciones curriculares, y los docentes dedican demasiado tiempo en crear material académico para estudiantes que lo necesiten: tienen que ajustar la fuente de texto, el tamaño, buscar imágenes en Internet o escanearlas de libros, etc., e incluso algunos tienen que hacerlo a mano.

Este proyecto consiste en el desarrollo de una aplicación web, *AdaptaMaterialEscolar*, creada específicamente para facilitar a los docentes la adaptación curricular no significativa de exámenes, actividades y unidades didácticas personalizadas para cada alumno. Para ello, hemos implementado un editor de texto que cuenta con los formatos y las funcionalidades más utilizadas por los profesores, con el fin de que puedan realizar las adaptaciones de manera más rápida, siguiendo un Diseño Centrado en el Usuario. Las funcionalidades que se han implementado en *AdaptaMaterialEscolar* son: un buscador de pictogramas, ejercicios de completar huecos en blanco, de generación de sopas de letras, de verdadero o falso, de definir conceptos, y de pregunta para desarrollar con un espacio limitado.

La aplicación web ha sido probada y evaluada por usuarios finales que pretenden usarla en el período escolar. Los resultados obtenidos demuestran que se cubre una necesidad real del profesorado y su interés por seguir desarrollando y mejorando *AdaptaMaterialEscolar*.

Palabras Clave

Adaptación curricular no significativa

Accesibilidad

Pictogramas

Herramienta docente

Editor de texto

Aplicación web

Material académico

Summary

By law, the educational curriculum seeks to ensure that the training process of students has the necessary materials, resources, and content during their academic journey, under equal conditions. However, since not all students have the same learning tools, it is possible to make curricular adaptations.

Currently, there are no free tools to make curricular adaptations, and teachers spend too much time creating academic material for students who need it: they have to adjust the text font, size, search for images on the Internet or scan them from books, etc., and some even have to do it by hand.

This project consists of the development of a web application, AdaptaMaterialEscolar, created specifically to facilitate teachers the non-significant curricular adaptation of exams, activities, and personalized teaching units for each student. For this purpose, we have implemented a text editor that has the formats and functionalities most used by teachers, so that they can make the adaptations more quickly, following a User-Centered Design. The functionalities that have been implemented in AdaptaMaterialEscolar are a pictogram finder and exercises to fill in the blanks, generate word searches, true or false, define concepts, and questions to develop with limited space.

The web application has been tested and evaluated by end-users who intend to use it in the school period. The results obtained show that a real need for teachers and their interest in continuing to develop and improve AdaptaMaterialEscolar is covered.

Key Words

Non-significant curriculum adaptation
Accessibility
Pictograms
Teaching tool
Text editor
Web application
Academic material

Índice

Agradecimientos	V
Resumen	VII
Palabras Clave	IX
Summary	XI
KeyWords	XIII
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura del proyecto	3
2. Introduction	5
2.1. Motivation	5
2.2. Objectives	6
2.3. Project structure	7
3. Estado del Arte	9
3.1. Adaptación Curricular	9
3.1.1. Lectura fácil para adaptación curricular	11
3.1.2. Pictogramas para adaptación curricular	13
3.2. Herramientas existentes para adaptaciones curriculares	15
3.3. Diseño Centrado en el Usuario	17
3.3.1. Conocer al usuario	19
3.3.2. Prototipado	20
3.3.3. Evaluación	22
4. Herramientas empleadas	23
4.1. Moqups	23
4.2. React	25

4.3. API de ARASAAC	27
4.4. Jest	28
4.5. CKEditor	31
4.6. Word-search	32
5. Metodología de desarrollo	37
5.1. Introducción	37
5.2. Kanban	38
5.3. Tipos de pruebas	41
5.3.1. Pruebas de memoria	41
5.3.2. Pruebas de implementación	41
6. AdaptaMaterialEscolar	45
6.1. Requisitos de la aplicación	45
6.1.1. Captura de requisitos	46
6.1.2. Análisis de requisitos	48
6.1.3. Requisitos adicionales	52
6.2. Diseño de la aplicación	52
6.2.1. Primera iteración: boceto inicial	53
6.2.2. Segunda iteración: versión 1.0 del prototipo	53
6.2.3. Tercera iteración: iteración competitiva	56
6.2.4. Cuarta iteración: diseño final	62
6.3. Implementación	62
6.3.1. Arquitectura	68
6.3.2. Funcionalidades	72
6.3.3. Estructura del código	97
7. Evaluación de AdaptaMaterialEscolar	101
7.1. Introducción	101
7.2. Diseño de la evaluación	102
7.3. Desarrollo de la evaluación	104
7.4. Resultado de la evaluación	105
7.5. Conclusión de la evaluación	108
8. Conclusiones y Trabajo futuro	109
8.1. Conclusiones	109
8.2. Trabajo futuro	111
9. Conclusions and Future Work	115
9.1. Conclusions	115
9.2. Future work	117

10.Trabajo Individual	121
10.1. Natalia	121
10.2. Jorge	123
10.3. Pablo	124
A. Tabla de puntuaciones individuales y examen de prueba	127
Bibliografía	131

Índice de figuras

3.1. Ejemplo de documento adaptado a lectura fácil. En este caso, un documento jurídico	12
3.2. Ejemplo de pictograma de una rosa	13
3.3. Ejemplo de pictograma que representa la acción de comer . . .	13
3.4. Ejemplo de pictogramas del Sistema Pictográfico de Comunicación (SPC)	14
3.5. Ejemplo de sistema pictográfico Minspeak	15
3.6. Interfaz de ARAWORD	16
3.7. Ventana de generación de ejercicio de asociar conceptos con EasyTestMaker	17
3.8. Ejemplo de examen generado con EasyTestMaker	17
3.9. Interfaz de Resoomer	18
3.10. Interfaz de Canva para la creación de un esquema usando una plantilla	18
3.11. Ejemplo de guión gráfico	20
3.12. Ejemplo de boceto en papel	21
3.13. Ejemplo de prototipo en papel interactivo	22
4.1. Interfaz de Moqups	24
4.2. Menú superior	24
4.3. Menú lateral izquierda	25
4.4. Menú lateral derecha (formato)	25
4.5. Menú lateral derecha (interacciones)	25
4.6. Funcionalidad sin usar JSX	26
4.7. Funcionalidad empleando JSX	26
4.8. Test pasado correctamente	30
4.9. Test no pasado correctamente	30
4.10. Ejemplo de CKEditor 5	32
4.11. Ejemplo de resultado de sopa de letras usando grid()	34
4.12. Ejemplo de resultado de sopa de letras usando toString() . . .	35
4.13. Ejemplo de las palabras insertadas al usar words()	35

4.14. Ejemplo del objeto resultante al usar <code>dump()</code>	36
4.15. Ejemplo de llamada a la función <code>read()</code>	36
4.16. Ejemplo del string formado cuando se usan los parámetros de la Figura 4.15	36
5.1. Tablero <i>Kanban</i>	40
6.1. Boceto inicial de la aplicación	53
6.2. Versión 1.0 de la página principal	54
6.3. Versión 1.0 del editor	55
6.4. Versión 1.0 de la búsqueda de pictogramas	56
6.5. Versión 1.0 de la adaptación de actividades	56
6.6. Versión 1.0 de la adaptación de temario	57
6.7. Prototipo de la página principal (Jorge)	58
6.8. Prototipo de la página principal (Natalia)	58
6.9. Prototipo de la página principal (Pablo)	59
6.10. Prototipo del editor (Jorge)	59
6.11. Prototipo del editor con un desplegable en uno de los menús (Jorge)	60
6.12. Prototipo de la vista previa (Jorge)	61
6.13. Prototipo del editor (Natalia)	61
6.14. Prototipo del editor (Pablo)	62
6.15. Página principal de <code>AdaptaMaterialEscolar</code>	63
6.16. Página principal de <code>AdaptaMaterialEscolar</code> tras subir un do- cumento	64
6.17. Interfaz de <code>AdaptaMaterialEscolar</code> al seleccionar Pictogramas	64
6.18. Interfaz de <code>AdaptaMaterialEscolar</code> al seleccionar Completar huecos	65
6.19. Interfaz de <code>AdaptaMaterialEscolar</code> al seleccionar Definiciones	65
6.20. Interfaz de <code>AdaptaMaterialEscolar</code> al seleccionar Desarrollo .	66
6.21. Interfaz de <code>AdaptaMaterialEscolar</code> al seleccionar Sopa de letras	66
6.22. Interfaz de <code>AdaptaMaterialEscolar</code> al seleccionar V/F	67
6.23. Ejemplo de ventana movable	67
6.24. Diagrama de flujo en Redux	68
6.25. Comparación de flujo de información de componentes sin y con Redux. Fuente: https://blog.codecentric.de/en/2017/12/developing-modern-offline-apps-reactjs-redux-electron-part-3-reactjs-redux-basics/	70
6.26. Diagrama de secuencia de subir un documento	73
6.27. Diagrama de flujo de subir un documento	73
6.28. Diagrama de secuencia del proceso de inicialización del editor y la ejecución de la función <code>execute()</code>	78

6.29. Resultados del buscador de pictogramas	79
6.30. Pictograma introducido en el editor	79
6.31. Diagrama de clases del buscador de pictogramas	80
6.32. Diagrama de secuencia del buscador de pictogramas	80
6.33. Vista previa de la sopa de letras	81
6.34. Diagrama de flujo de la generación de la sopa de letras	83
6.35. Diagrama de secuencia de la generación de la sopa de letras	84
6.36. Resultado de la generación de la sopa de letras	85
6.37. Diagrama de flujo cuando se modifica el texto de una definición	86
6.38. Diagrama de secuencia cuando se modifica el texto de una definición	86
6.39. Ejemplo I de opciones del ejercicio de definiciones	87
6.40. Resultado del ejercicio de definiciones	88
6.41. Diagrama de flujo cuando se modifica el enunciado del ejercicio pregunta para desarrollar	89
6.42. Diagrama de secuencia cuando se modifica el enunciado del ejercicio pregunta para desarrollar	89
6.43. Ejemplo de opciones del ejercicio pregunta para desarrollar	90
6.44. Resultado de ejercicio pregunta para desarrollar	90
6.45. Diagrama de flujo cuando añaden más filas al ejercicio de verdadero o falso	91
6.46. Diagrama de secuencia cuando añaden más filas al ejercicio de verdadero o falso	92
6.47. Ejemplo de opciones del ejercicio verdadero o falso	92
6.48. Resultado de ejercicio verdadero o falso	93
6.49. Ejemplo de edición del texto del ejercicio completar huecos	93
6.50. Ejemplo de selección de palabras del ejercicio completar huecos	94
6.51. Diagrama de secuencia cuando se hace clic sobre Seleccionar palabras en el ejercicio completar huecos	95
6.52. Diagrama de flujo al seleccionar una palabra del ejercicio completar huecos	95
6.53. Diagrama de secuencia cuando se hace clic en una palabra en el ejercicio completar huecos	96
6.54. Resultado del ejercicio de completar huecos	97
6.55. Ejemplo de organización de Components	98
6.56. Ejemplo de organización de Pages	99
6.57. Ejemplo de organización de Redux	99
6.58. Ejemplo de organización de CKEditor	100
7.1. Gráfica de resultados de la Escala de Usabilidad de un Sistema	106
7.2. Gráfica de resultados de Diseño, Funcionalidades y Utilidad real	107

Índice de Tablas

6.1. Puntuación por importancia (Imp.) y dificultad (Dif.) de todos los requisitos de la aplicación	50
6.2. Lista de funcionalidades relacionadas con adaptaciones de temario (T) y ejercicios (E), ordenada por prioridad	51
6.3. Lista de funcionalidades de adaptación de formato, ordenada por prioridad	51
7.1. Cálculo del resultado del cuestionario de SUS	106
A.1. Puntuación individual de cada miembro del equipo de desarrollo para cada requisito	128

Capítulo 1

Introducción

RESUMEN: En este capítulo se realiza la introducción del Trabajo de Fin de Grado que va a ser presentado en este documento. Primero, en la Sección 1.1, se explicará la motivación que ha dado lugar al trabajo. A continuación, en la Sección 1.2, el objetivo que se pretende alcanzar. Por último, la estructura del proyecto final, en la Sección 1.3.

1.1. Motivación

La educación escolar tiene como finalidad promover el desarrollo de ciertas capacidades y el aprendizaje de determinados contenidos necesarios para que los alumnos puedan ser miembros activos de la sociedad. Para ello, la escuela debe ofrecer una respuesta educativa que evite la discriminación y asegure la igualdad de oportunidades.

El sistema educativo español, en la actualidad, está organizado en ocho etapas o niveles que garantizan el derecho a una educación inclusiva para el alumnado en cada fase de su desarrollo cognitivo y emocional (González, 2020): Educación infantil, Educación primaria, ESO, Bachillerato, Formación profesional, Carrera universitaria, Másteres y postgrados y Doctorado. De éstas, únicamente dos son obligatorias: Educación Primaria (EP) y Educación Secundaria Obligatoria (ESO).

En España hay aproximadamente unos 2 millones de alumnos con necesidades educativas especiales (NEE), aunque no todos ellos presentan algún tipo de discapacidad, frente a los 8 millones de estudiantes que hay en enseñanzas de régimen general no universitarias.

Los materiales, recursos y contenidos que se presentan y usan en los centros educativos están regulados por ley en el currículum educativo o escolar. El currículum educativo trata de garantizar que todos los alumnos terminan cada curso igual de preparados y regula los elementos que determinan los

procesos de enseñanza y aprendizaje de cada una de las asignaturas, incluyendo: los objetivos de cada enseñanza y etapa educativa, las competencias y contenidos, la metodología didáctica, los estándares y resultados de aprendizaje y los criterios de evaluación.

En el currículo escolar existen unas necesidades educativas comunes, compartidas por todos los alumnos. Sin embargo, no todos los estudiantes se enfrentan con las mismas herramientas al aprendizaje, cada estudiante tiene una necesidad individual. La mayoría de éstas son resueltas a través de actuaciones “sencillas”: dar más tiempo al alumno para el aprendizaje de determinados contenidos, diseñar actividades complementarias... Sin embargo, existen necesidades individuales que no pueden ser resueltas por estos medios, siendo necesarias una serie de medidas pedagógicas especiales distintas de las que requieren habitualmente la mayoría de los alumnos. En este caso se habla de necesidades educativas especiales, y para atender estas necesidades son necesarias adaptaciones curriculares. Existen dos tipos de adaptaciones curriculares:

- **Adaptación no significativa:** no se modifican los contenidos curriculares de las asignaturas, sino que se adaptan los materiales, exámenes... Los encargados de realizar estas adaptaciones son los profesores.
- **Adaptación significativa:** se eliminan apartados del currículo oficial.

Los docentes dedican demasiado tiempo a la creación del material académico de los estudiantes que necesitan adaptaciones no significativas. Entre otras cosas, tienen que ajustar la fuente del texto y el tamaño, buscar imágenes en Internet o escanearlas de los libros, redactar resúmenes resaltando la información más relevante, etc. Este TFG pretende proporcionar una herramienta al profesorado que facilite la adaptación de los contenidos curriculares de las asignaturas, reduciendo de esta forma el tiempo y esfuerzo que deben dedicar los docentes a estas tareas.

1.2. Objetivos

El objetivo de este TFG es desarrollar una herramienta de trabajo para el profesorado que permita adaptar los recursos de las asignaturas en cualquier formato de manera intuitiva, fácil y rápida, y así poder crear unidades didácticas personalizadas que se ajusten a las necesidades de cada estudiante.

La herramienta que resulte de este TFG permitirá crear material escolar personalizado ajustado para cada alumno, permitiendo cambios de formato, la creación de diferentes tipos de ejercicios (como por ejemplo, rellenar espacios en blanco, sopas de letras o ejercicios de desarrollo) y un buscador integrado de pictogramas. Estará basada en una Arquitectura Redux, en la que se separarán internamente las vistas de los modelos de datos. Además, se

recurrirán a servicios externos que proporcionarán funcionalidad a nuestra aplicación.

Además, para que la aplicación se adapte a las necesidades reales de nuestros usuarios finales (los docentes), se va seguir un Diseño Centrado en el Usuario (DCU). Para ello, realizaremos entrevistas a los usuarios finales con el fin de definir qué necesidades tienen, qué diseño se ajusta más a sus requisitos y qué funcionalidades tienen mayor prioridad. También contaremos con ellos para evaluar la herramienta.

En cuanto a los objetivos académicos, nuestra meta principal es aplicar, en un proyecto real, los conocimientos adquiridos durante el Grado de Ingeniería del Software y ampliarlos.

1.3. Estructura del proyecto

La estructura del proyecto está formada por diez capítulos, incluyendo este introductorio, en el que se explica la motivación y el objetivo del trabajo. A continuación, se expone el resumen de cada uno de ellos:

- En el **capítulo dos** se presenta la introducción, traducida al inglés.
- En el **capítulo tres** se presenta el estado del arte, donde se explicará el dominio en el que se enmarca el proyecto.
- En el **capítulo cuatro** se describen las herramientas utilizadas en el desarrollo del proyecto.
- En el **capítulo cinco** se explica la metodología de desarrollo que ha sido empleada durante todo el proceso y el motivo por el que se escogió dicha metodología.
- En el **capítulo seis** se explican las fases de captura de requisitos, diseño e implementación de la aplicación.
- En el **capítulo siete** se muestra cómo se ha hecho la evaluación de AdaptaMaterialEscolar.
- En el **capítulo ocho** se describen las conclusiones y el trabajo futuro.
- En el **capítulo nueve** se describen las mismas conclusiones y trabajo futuro, traducidas al inglés.
- En el **capítulo diez** se presenta el trabajo individual que ha realizado cada miembro del equipo en el proyecto.

Capítulo 2

Introduction

RESUMEN: This chapter introduces the Final Degree Project that will be presented in this document. First, in Section 2.1, the motivation that lead to this project will be explained. Then, in Section 2.2, the objective to be achieved. Finally, the structure of the final project, in Section 2.3.

2.1. Motivation

The purpose of school education is to promote the development of certain skills and the learning of certain contents necessary for students to become active members of society. To achieve this, the school must offer an educational response that avoids discrimination and ensures equal opportunities.

Currently, the Spanish education system is organized into eight stages or levels that guarantee the right to inclusive education for students at every stage of their cognitive and emotional development (González, 2020): Pre-school education, Primary education, Secondary education, High school, Vocational training, University studies, Master and postgraduate degrees, and Doctorates. Of these, only two are compulsory: Primary Education (PE) and Compulsory Secondary Education (ESO).

In Spain, there are approximately 2 million students with special educational needs (SEN), although not all of them have some type of disability, compared to the 8 million students in non-university general education.

The materials, resources, and content that are presented and used in educational centers are regulated by law in the educational or school curriculum. The educational curriculum tries to guarantee that all students finish each course equally prepared and regulates the elements that determine the teaching and learning processes of each of the subjects, including the objectives of each teaching and educational stage, the competencies and contents, the

didactic methodology, the learning standards, and results and the evaluation criteria.

In the school curriculum, there are common educational needs, shared by all the students. However, not all students face learning with the same tools; each student has an individual need. Most of the individual needs of students are solved through “simple” actions: giving the student more time to learn certain contents, designing complementary activities... However, there are individual needs that cannot be solved by these means, requiring a series of special pedagogical measures different from those usually required by most students. In this case, we speak of special educational needs, and to meet these needs curricular adaptations are necessary. There are two types of curricular adaptations:

- **Non-significant adaptation:** the curricular contents of the subjects are not modified, but other resources like the materials or exams are adapted. The teachers will be in charge of making these adaptations.
- **Significant adaptation:** sections of the official curriculum are eliminated.

Teachers spend too much time creating academic material for students who need non-significant adaptations. Among other things, they have to adjust the text font and size, search for images on the Internet or scan them from books, write summaries highlighting the most relevant information, etc. This FDP aims to provide a tool for teachers to facilitate the adaptation of the curricular contents of the subjects, thus reducing the time and effort that teachers must devote to these tasks.

2.2. Objectives

The objective of this TFG is to develop a working tool for teachers that allows them to adapt the resources of the subjects in any format in an intuitive, easy, and fast way, and thus be able to create customized didactic units that fit the needs of each student.

The tool resulting from this TFG will allow the creation of customized school material adjusted to each student, allowing format changes, the creation of different types of exercises (such as fill-in-the-blank, word searches, or development exercises), and an integrated pictogram search engine. It will be based on a Redux Architecture, in which the views of the data models will be separated internally. In addition, external services will be used to provide the functionality to our application.

Furthermore, to adapt the application to the real needs of our end users (the teachers), we will follow a User-Centered Design (UCD). To do this, we will conduct interviews with end-users to define their needs, which design

best suits their needs, and which functionalities have the highest priority. We will also count on them to evaluate the tool.

As for the academic objectives, our main goal is to apply, in a real project, the knowledge acquired during the Software Engineering Degree and to extend it.

2.3. Project structure

The project's structure consists of ten chapters, including this introductory chapter, which explains the motivation and objective of the work translated into English. The following is a summary of each chapter:

- **Chapter one** presents the introduction in Spanish.
- **Chapter three** presents the state of the art, where the domain in which the project is framed will be explained.
- **Chapter four** describes the tools used in the development of the project.
- **Chapter five** explains the development methodology that has been used during the whole process and the reason why this methodology was chosen.
- **Chapter six** explains the phases of requirements capture, design, and implementation of the application.
- **Chapter seven** shows how the evaluation of AdaptaMaterialEscolar has been done.
- **Chapter eight** describes the conclusions and future work.
- **Chapter nine** describes the same conclusions and future work, translated into English.
- **Chapter ten** presents the individual work that each team member has done on the project.

Capítulo 3

Estado del Arte

RESUMEN: En este capítulo nos centraremos en explicar el dominio en el que se enmarca nuestro proyecto. En la sección 3.1 explicaremos en detalle las necesidades de las personas con necesidades especiales, y cómo se tratan de satisfacer mediante adaptaciones curriculares. Además, en la sección 3.2, mencionaremos algunas de las herramientas actuales para realizar adaptaciones curriculares orientadas a nuestro objetivo. Por otra parte, se explicarán en la sección 3.3 las bases del Diseño Centrado en el Usuario de cara a ofrecer una experiencia de usuario satisfactoria.

3.1. Adaptación Curricular

Los materiales, recursos y contenidos que se presentan y usan en los centros educativos están regulados por ley en el currículo educativo o escolar. El currículo educativo trata de garantizar que todos los alumnos terminan cada curso igual de preparados, e incluye todos los recursos académicos, materiales y humanos necesarios para llevar a cabo el proyecto educativo marcado por la legislación.

Las principales funciones del currículo educativo son: determinar las asignaturas, contenidos y temáticas comunes a todos los alumnos, determinar los criterios de evaluación y logros que debe superar el alumnado y formalizar los estándares educativos que permitan cumplir unos objetivos establecidos.

En el currículo escolar existen unas necesidades educativas comunes, com-

partidas por todos los alumnos. Sin embargo, no todos los estudiantes se enfrentan con las mismas herramientas al aprendizaje, cada estudiante tiene una necesidad individual. La mayoría de las necesidades individuales de los estudiantes son resueltas a través de actuaciones “sencillas”: dar más tiempo al alumno para el aprendizaje de determinados contenidos, diseñar actividades complementarias... Sin embargo, existen necesidades individuales que no pueden ser resueltas por estos medios, siendo necesarias una serie de medidas pedagógicas especiales distintas de las que requieren habitualmente la mayoría de los alumnos. En este caso se habla de necesidades educativas especiales. Para atender estas necesidades son necesarias adaptaciones curriculares. La adaptación curricular (o adecuación curricular) es la modificación de elementos del currículo a fin de dar respuesta a las necesidades especiales del alumnado. Las adaptaciones curriculares se clasifican en (CADAH, 2012):

- **Adaptaciones Curriculares de Acceso al Currículo.** Responden a las necesidades de un grupo de personas. Estos pueden ser:
 - **De Acceso Físico.** Representa a los recursos materiales y espaciales. Ejemplos de estos son mobiliario adaptado, iluminación y sonoridad adaptada, o profesorado de apoyo especializado en metodologías que faciliten el aprendizaje de los alumnos.
 - **De Acceso a la Comunicación.** Incluye materiales específicos de enseñanza tales como: aprendizaje, ayudas tecnológicas o sistemas de computación complementarios. Algunos ejemplos son el braille, la lengua de signos o la comunicación a través del ordenador.
- **Adaptaciones Curriculares Individualizadas.** Se realizan para un único alumno con el fin de responder a necesidades educativas especiales que no pueden ser compartidas por el resto de los compañeros. Pueden ser:
 - **No Significativas.** Adaptan materiales, tiempos, actividades, metodologías, técnicas e instrumentos de evaluación sin modificar los contenidos curriculares de las asignaturas. Es la estrategia principal para conseguir la individualización de la enseñanza y por tanto, tienen un carácter preventivo y compensador.
 - **Significativas.** Modificaciones que se realizan desde la programación, previa evaluación psicopedagógica, y que afectan a los elementos prescriptivos del currículo oficial a modificar e incluso eliminar objetivos generales de la etapa, contenidos básicos y nucleares de las diferentes áreas curriculares y criterios de evaluación.

Aunque la adaptación curricular incluye un amplio número de posibilidades y técnicas, en general es necesario simplificar el lenguaje que se utiliza, e incluso ofrecer alternativas al mismo. De ambas cuestiones se encargan, entre otros, la lectura fácil y los pictogramas, que se explicarán a continuación.

3.1.1. Lectura fácil para adaptación curricular

No todas las personas tienen la misma capacidad de entender el lenguaje natural, ya que esta capacidad puede estar limitada por diferentes causas. Según el manual “Lectura fácil: Métodos de redacción y evaluación” (Óscar García Muñoz, 2012), las personas más afectadas por este problema son las personas con discapacidad intelectual. Algunos ejemplos son las personas con Síndrome de Down o las personas con enfermedades y trastornos mentales y del comportamiento, como las personas con Trastorno del Espectro Autista (TEA). También se ven afectadas las personas con dificultad para el desarrollo del lenguaje por discapacidad auditiva y las personas con circunstancias transitorias de dificultad en la comprensión lectora. Por ello, las adaptaciones a lectura fácil pueden suponer un gran beneficio para estos usuarios. Las adaptaciones curriculares para estos casos requieren documentos con un lenguaje más claro y comprensible, y una estructura más simple y esquemática. Las directrices de Lectura Fácil establecen un gran número de reglas, tanto estéticas como de redacción, para poder facilitar estas adaptaciones y garantizar un texto de fácil lectura. Algunas de las medidas que resultan de interés para este trabajo son:

- **Gramática.** Evitar ciertos tiempos verbales, como el futuro o el subjuntivo, o usar oraciones simples y cortas, son algunas de las medidas gramaticales que propone el manual.
- **Ortografía.** Tener en cuenta ciertas pautas para evitar el uso de signos ortográficos que puedan dificultar la comprensión lectora. Se recomienda evitar el punto y seguido, y usar el punto y aparte para así dejar bien diferenciadas las frases. También se aconseja evitar el punto y coma y los puntos suspensivos. En cuanto a los números, es preferible que se escriban en formato cifra.
- **Léxico.** Tratar de usar un vocabulario que no sea complejo. Se aconseja utilizar palabras sencillas expresadas de forma simple y que, en la medida de lo posible, no sean largas. También se recomienda utilizar siempre el mismo sinónimo y evitar abreviaturas y siglas.
- **Tipografía.** El tamaño de letra debe de ser lo suficientemente grande, entre 12 y 16 puntos, y se deben utilizar tipografías sin remate como

Arial o Helvetica y reforzar la nitidez de los números.

- **Composición del texto.** Para las líneas, recomienda que cada una esté compuesta de una sola oración y que no superen los 60 caracteres. Contendrán un mínimo de 5 palabras y un máximo de 15 a 20, de modo que no queden ni muy cortas ni muy largas. En cuanto a los párrafos, aconseja alinear el texto a la izquierda, en párrafos y capítulos cortos, de forma que se favorezcan las pausas frecuentes. Recomienda que se mantenga un ritmo regular en la composición de párrafos para que así se muestre de una forma más nítida y organizada visualmente. Respecto a la distribución del espacio, recomienda una distribución ordenada y poco densa, cuanto más blanco mejor, y evitar un diseño en columnas.
- **Imágenes.** Se recomienda utilizar imágenes de apoyo al texto que estén bien referenciadas y utilizar símbolos o dibujos para ideas, conceptos o temas abstractos. Uno de los principales tipos de imágenes de apoyo son los pictogramas, que se explicarán en el siguiente apartado.

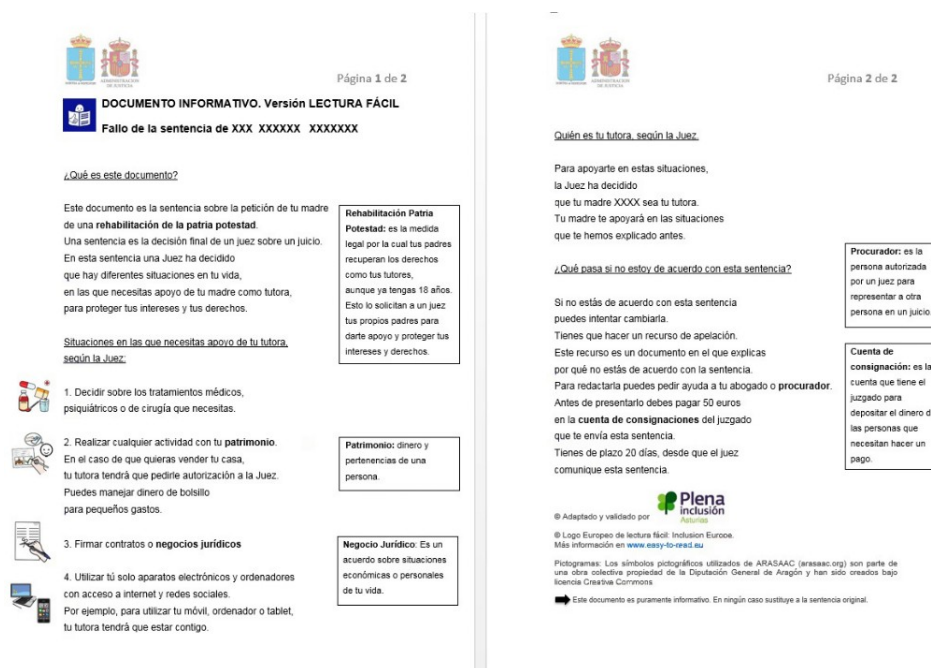


Figura 3.1: Ejemplo de documento adaptado a lectura fácil. En este caso, un documento jurídico

En la Figura 3.1 se muestra un ejemplo de un texto adaptado a lectura fácil. En este texto se puede ver un documento jurídico adaptado, en el que se aplican medidas mencionadas previamente, como la mayor separación entre líneas y el uso de pictogramas, además de usar un lenguaje más sencillo, con frases más cortas y simples.

3.1.2. Pictogramas para adaptación curricular

Las personas con carencias sensoriales, cognitivas o con un conocimiento insuficiente de la lengua pueden presentar problemas a la hora de comunicarse. Por ello, los Sistemas Aumentativos y Alternativos de Comunicación (SAACs) (Basil, 2017), que se encargan de proporcionar información en medios alternativos, o aumentando éste, se presentan como una gran opción para la comunicación de estas personas. Uno de los SAACs más comunes son los pictogramas.

Un pictograma (Logopedia, 2012) es un signo claro y esquemático que representa un objeto real, figura o concepto, sintetizando un mensaje que puede señalar o informar sobrepasando la barrera de las lenguas. Es un recurso comunicativo de carácter visual que podemos encontrar en diversos contextos de nuestra vida diaria y nos aporta información útil conocida por todos. Los pictogramas son:

- **Universales.** Pueden ser entendidos por cualquier persona, independientemente de su idioma.
- **Visuales.** Muestran con claridad y sencillez al objeto/acción que representan.
- **Inmediatos.** La comunicación se establece entre el emisor y el receptor tan solo señalando sobre el pictograma adecuado.

Las Figuras 3.2 y 3.3 son ejemplos de pictogramas. En el primero se muestra un objeto físico, en este caso una rosa, y en el segundo se representa la acción de comer.

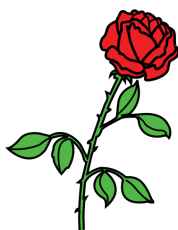


Figura 3.2: Ejemplo de pictograma de una rosa

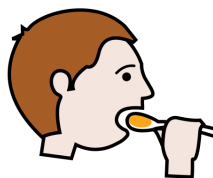


Figura 3.3: Ejemplo de pictograma que representa la acción de comer

Los sistemas pictográficos son sistemas que representan la realidad mediante pictogramas. Algunos ejemplos de sistemas pictográficos son:

- **Sistema Pictográfico de Comunicación (SPC) (Belloch, 2014).** Sus iconos representan de forma clara el concepto que pretenden transmitir. Son dibujos fácilmente diferenciables entre sí y de sencilla comprensión. La Figura 3.4 muestra un ejemplo de SPC, donde se ve una serie de pictogramas diferenciados por colores, junto con el significado de cada pictograma. Cada color representa una categoría diferente:

- **Amarillo:** Se refiere a personas, incluyendo pronombres personales y posesivos.
- **Verde:** Se refiere a verbos.
- **Azul:** Se refiere a descripciones, principalmente adjetivos y adverbios.
- **Rosa:** Se refiere a palabras de uso en interacciones sociales.
- **Blanco:** Se refiere a la parte miscelánea, es decir, artículos, preposiciones, conjunciones..., todo lo que no pertenece a las anteriores categorías.

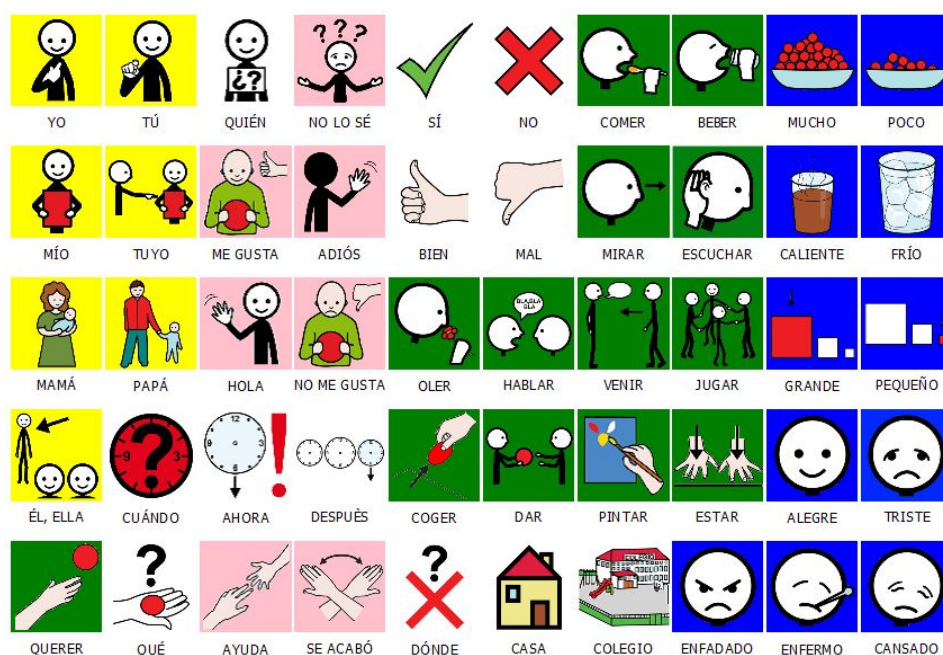


Figura 3.4: Ejemplo de pictogramas del Sistema Pictográfico de Comunicación (SPC)

- **Minspeak (Belloch, 2014).** En este sistema los pictogramas no tienen un significado concreto preestablecido, sino que se fija por el logopeda y usuario, lo que permite personalizar los mensajes. La Figura



Figura 3.5: Ejemplo de sistema pictográfico Minspeak

3.5 muestra un ejemplo de un sistema Minspeak, donde se define el significado de los pictogramas a través de una combinación de estos.

- **ARASAAC**¹. El Portal Aragonés de Comunicación Aumentativa y Alternativa (ARASAAC), creado en 2007, ofrece un amplio catálogo de pictogramas de libre acceso. Dichos pictogramas se encuentran divididos en categorías: Alimentación, Ocio, Lugar, Ser vivo, Educación, Tiempo, Miscelánea, Desplazamiento, Religión, Trabajo, Comunicación, Documento, Conocimiento y Objeto. Es uno de los sistemas pictográficos más usados en España, ya que incluye un amplio número de herramientas dedicadas a pictogramas, como buscadores o generador de documentos con pictogramas, como pueden ser calendarios u horarios. El catálogo de ARASAAC ofrece pictogramas tanto en color, como con blanco y negro, y fondos de colores. Los pictogramas de las Figuras 3.2 y 3.3 pertenecen a ARASAAC.

3.2. Herramientas existentes para adaptaciones curriculares

La existencia de herramientas que faciliten adaptaciones curriculares es muy limitada. Existen herramientas enfocadas a realizar solo alguna de las adaptaciones que se pueden necesitar, pero de manera individualizada.

*ARAWORD*² es una herramienta de procesamiento de textos que combina la escritura de texto con pictogramas y facilita la elaboración de materiales. También resulta una herramienta muy útil para ser usada por usuarios que están adquiriendo el proceso de la lectoescritura, ya que la aparición del pictograma, a la vez que se escribe, es un refuerzo muy positivo para reconocer y evaluar que la palabra o la frase escrita es correcta. En la Figura 3.6 se

¹<https://arasaac.org/>

²http://aulaabierta.arasaac.org/araword_inicio

muestra un ejemplo de la interfaz de ARAWORD, donde se traducen las oraciones a pictogramas.

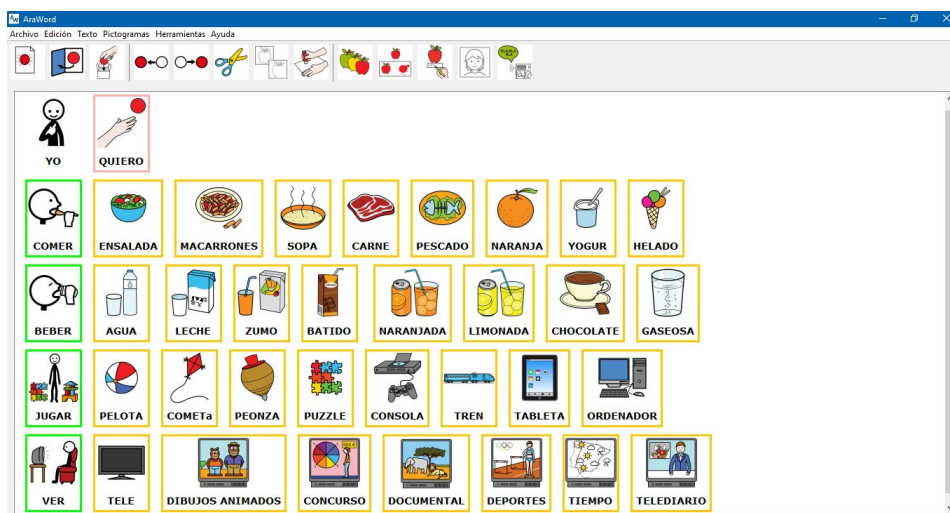


Figura 3.6: Interfaz de ARAWORD

*EasyTestMaker*³ es una web de creación de exámenes online. Dicha herramienta incluye opciones de generación de ejercicios sencillos para exámenes, como ejercicios de verdadero/falso, asociación de conceptos, rellenar espacios en blanco y respuestas cortas. La Figura 3.7 muestra un ejemplo de cómo se genera un ejercicio de asociar conceptos, mientras que la Figura 3.8 muestra el resultado con otros tipos de ejercicios. EasyTestMaker ofrece un plan gratuito que permite crear hasta 25 tests, mientras que la versión de pago ofrece más opciones como insertar imágenes, exportar a PDF y personalización del texto.

Por otra parte, existen herramientas que ayudan a la creación de esquemas y resúmenes. Una herramienta para realizar resúmenes es *Resoomer*⁴, la cual devuelve un texto resumido a través de un texto introducido. En la Figura 3.9 se muestra un ejemplo de la interfaz de Resoomer.

Para la generación de esquemas, habitualmente se usan herramientas de diseño como *Canva*⁵. Canva ofrece plantillas prediseñadas donde el usuario puede introducir las palabras a organizar. La Figura 3.10 muestra un ejemplo de cómo se hace un esquema con Canva.

³<https://www.easytestmaker.com/>

⁴<https://resoomer.com/es/>

⁵<https://www.canva.com/>

Matching [Instructions](#) [Image](#) [Options](#)

Enter options on left and answer on the right of the same row. Choose your ordering or click to randomize.

	1	2	3	4	
Cielo	1	Azul	4		✗
Tierra	2	Marrón	2		✗
Nube	3	Blanco	1		✗
	4				✗

[+ add option](#)

Point Value ☐ Bonus question

1 1.00 - 99.99

Point value is for each option. Total points for all options is the number of options times the point value.

[Ok - Randomize Order](#) [Ok - Keep Order](#) [Cancel](#)

Figura 3.7: Ventana de generación de ejercicio de asociar conceptos con EasyTestMaker

- 1) Los humanos están compuestos de células procariotas
 - a) True
 - b) False
- 2) _____ Cielo
- 3) _____ Tierra
- 4) _____ Nube
- a) Blanco
- b) Azul
- c) Marrón
- 5) La capital de Andalucía es _____, mientras que la capital de _____ es Barcelona.

Figura 3.8: Ejemplo de examen generado con EasyTestMaker

3.3. Diseño Centrado en el Usuario

El Diseño Centrado en el Usuario (DCU) (Moreno, Jiménez y Sánchez, 2019) consiste en un enfoque de diseño de interfaces cuyo objetivo es crear un producto que resuelva las necesidades de los usuarios finales, consiguiendo así una mayor calidad de producto y, por tanto, una mayor satisfacción

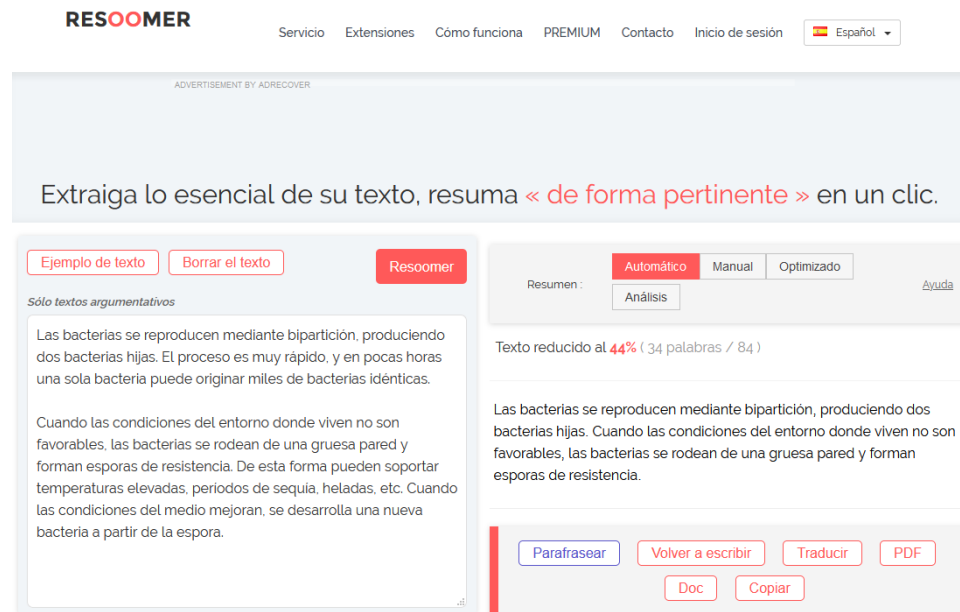


Figura 3.9: Interfaz de Resoomer

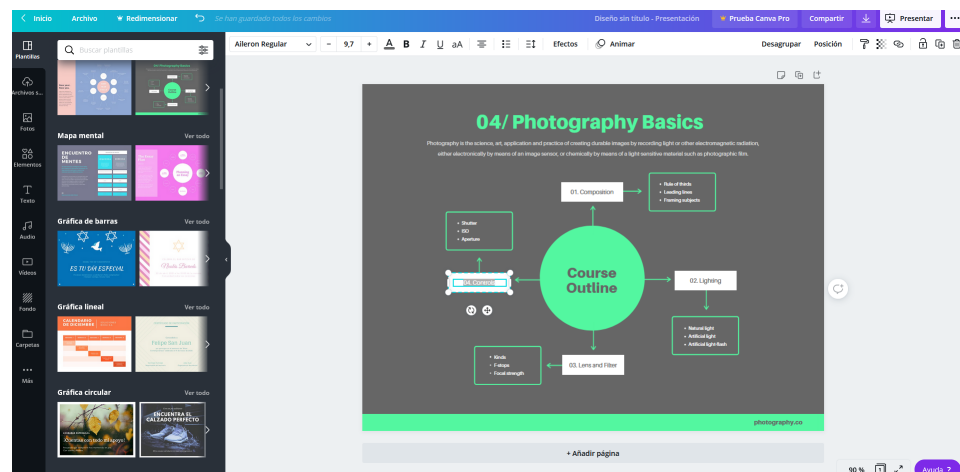


Figura 3.10: Interfaz de Canva para la creación de un esquema usando una plantilla

de los mismos. Con dicha metodología se pretende entender previamente a los usuarios para poder encontrar soluciones acordes a sus necesidades específicas. Este proceso se basa en tres pilares fundamentales: conocer al usuario, prototipado del producto y evaluación, que se explicarán en las siguientes secciones.

3.3.1. Conocer al usuario

Entender a los usuarios finales, el entorno y el uso que puedan darle al producto ayuda, a diseñar productos que puedan encajar de mejor forma en dicho contexto y mejorar la forma en la que trabajan, comunican e interactúan. Por ello, se realizan estudios en los que se planea obtener información para conocer quiénes van a ser los usuarios reales del sistema, cómo se comportan y cuál es el contexto en el que se desenvuelven.

Algunas de las técnicas más usadas para poder investigar cuáles son las necesidades del usuario son:

- **Entrevistas.** Se realiza un conjunto de preguntas al sujeto de estudio con el fin de aportar la información necesaria para poder comenzar con el diseño. Se suelen realizar diferentes tipos de preguntas, como las orientadas a objetivos, que pretenden descubrir las prioridades del usuario, orientadas al sistema, que identifican las funcionalidades más usadas, orientadas a flujos de trabajo, que identifican procesos y recurrencia de actividades, y las orientadas a actitudes de los usuarios finales.
- **Observación del usuario.** Analizar el comportamiento del usuario mientras realiza una actividad relacionada con el problema a resolver, para poder detectar si las necesidades reales coinciden con las necesidades descritas por el usuario.
- **Diarios de uso.** Se proporciona a los usuarios un diario en el que escriben información, como interacciones con el sistema, necesidades o un registro de comportamientos que engloban informaciones del usuario. Esta información servirá para conocer más a fondo lo que busca el usuario.
- **Análisis de la competencia.** Estudiar cómo los usuarios usan un sistema ya existente para poder descubrir qué características del producto son útiles y cuáles son los problemas que los usuarios encuentran en estos.

Gracias a estas técnicas se podrán definir de manera más precisa los requisitos del sistema para posteriormente proceder al prototipado de la aplicación.

3.3.2. Prototipado

Es importante tener clara cuál va a ser la interacción del usuario con el producto. Por ello, se hacen prototipos que simulan el diseño y comportamiento del sistema para que los clientes finales puedan dar su “feedback” y confirmen si es lo que realmente buscan. Dichos prototipos pueden ser realizados en papel o en medios digitales. Los prototipos en papel pueden ser de distintos tipos:

- **Guiones gráficos.** Conjunto de ilustraciones pintadas en forma de cómic, en el que se muestra un escenario y una secuencia de acciones que simulan casos de uso de la aplicación. En la Figura 3.11 se muestra un ejemplo de guión gráfico que secuencia una situación en la que una aplicación para controlar eventualmente el estado de ánimo de una persona cercana pudiera ser útil.



Figura 3.11: Ejemplo de guión gráfico

- **Bocetos.** Diseños de interfaces dibujados a mano en papel. Se enfocan más en aspectos genéricos y de alto nivel, ignorando los detalles. En la Figura 3.12 se muestra un ejemplo de un boceto en papel de una aplicación para móvil.
- **Prototipos en papel interactivos.** Es una maqueta o “mockup” de la interfaz hecha en papel y compuesta de las piezas que representan a

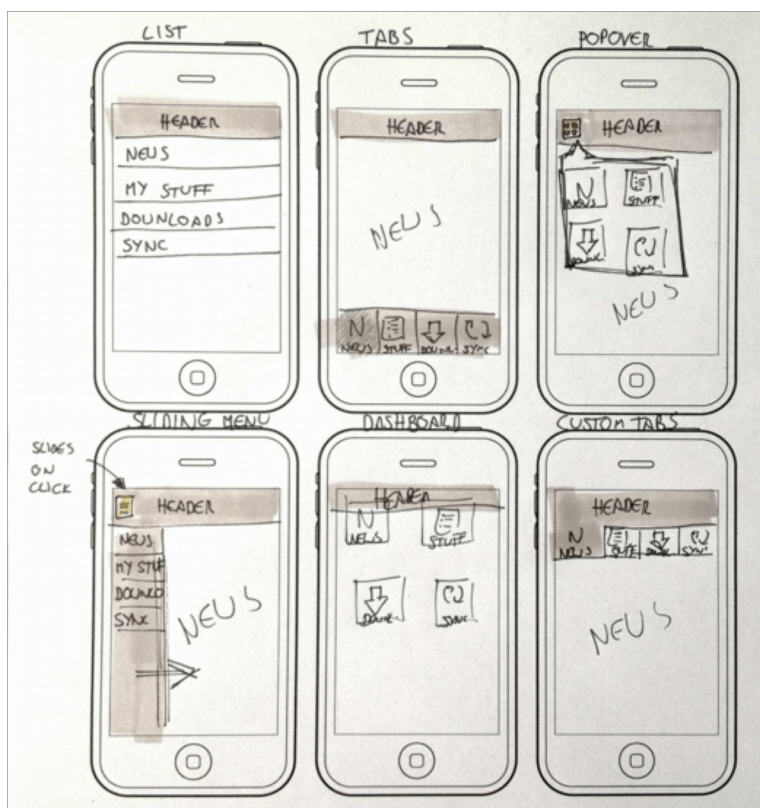


Figura 3.12: Ejemplo de boceto en papel

la aplicación. Estos prototipos son interactivos, es decir, muestran una simulación del comportamiento del sistema ante acciones del usuario. Algunas formas de representar estas interacciones son mostrar cambios en la interfaz a raíz de acciones realizadas (como clicks o pulsaciones de botones del teclado). La Figura 3.13 muestra un ejemplo de prototipo en papel interactivo en el que se muestran los cambios al realizar acciones sobre la aplicación.

Los prototipos digitales suelen ser mucho más fieles al diseño final del sistema. Además, permiten diseñar interacciones de manera más realista. Algunas herramientas para realizar prototipos digitales son *Balsamiq*⁶, *JustInMind*⁷, *Moqups*⁸, o incluso HTML y CSS y Photoshop.

⁶<https://balsamiq.com/>

⁷<https://www.justinmind.com/>

⁸<https://moqups.com/>

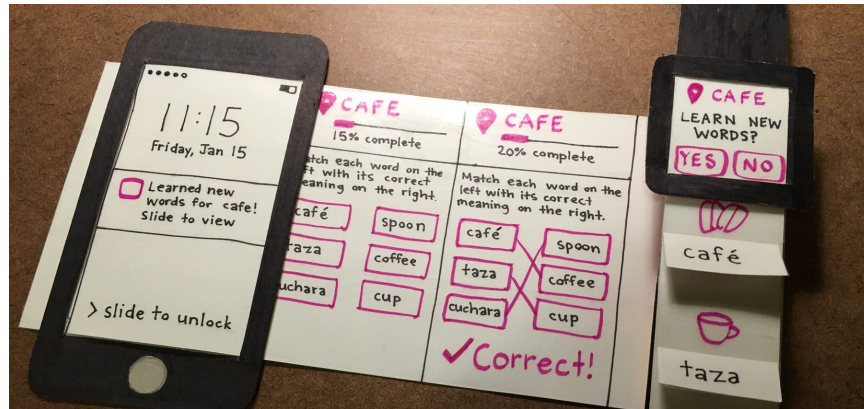


Figura 3.13: Ejemplo de prototipo en papel interactivo

3.3.3. Evaluación

Una interfaz tiene que ser probada y evaluada por el usuario final para garantizar que responde a las expectativas o detectar problemas de usabilidad. Existen dos tipos de evaluaciones, las heurísticas (con expertos) y las que se realizan con usuarios:

- **Evaluación heurística.** Identifica los problemas de usabilidad siguiendo directrices universales. Estas evaluaciones suelen realizarse por expertos en Experiencia de Usuario y facilitan la identificación temprana de problemas de usabilidad.
- **Evaluación con usuarios.** Simula ejecuciones del sistema realizadas por los usuarios finales. Los usuarios interactúan con el programa y observan hasta qué punto son capaces de realizar determinados conjuntos de tareas. Además, se obtiene un “feedback” completo del proceso mediante un plan de evaluación que incluye, entre otros, preguntas a los usuarios.

Todo este proceso de diseño muestra un carácter iterativo. El usuario final se verá involucrado de manera constante con el fin de dar feedback y garantizar que el prototipo diseñado cumple con lo esperado.

Capítulo 4

Herramientas empleadas

RESUMEN: En este capítulo se explican las herramientas y librerías que se han empleado en el desarrollo de este proyecto. En la sección 4.1 se introduce la herramienta online Moqups. En la sección 4.2 se muestra el *framework* que se ha utilizado para la creación de la página web. En la sección 4.3 se habla de la API de ARASAAC usada para la búsqueda de pictogramas. En la sección 4.4 se presenta la librería *Jest* usada para la realización de pruebas. En la sección 4.5 se explica CKEditor, el editor de texto que hemos usado en la aplicación. En la sección 4.6 se explica el paquete NPM usado para la generación de las sopas de letras.

4.1. Moqups

Moqups¹ es una página web para la creación de bocetos, prototipos, diagramas, etc. Permite diseñar una interfaz, o simplemente ver cómo es el flujo de un algoritmo, arrastrando desde los menús al espacio de trabajo los distintos elementos, llamados plantillas, que podemos encontrar en una aplicación (barras de progreso, etiquetas o *labels*, enlaces, diferentes tipos de ventanas, etc.).

Existe la posibilidad de añadir comentarios e iconos, y poder crear diferentes páginas en un mismo proyecto (por ejemplo, crear varias vistas de una aplicación), así como añadir interacción a los elementos (como por ejemplo,

¹<https://moqups.com/>

haciendo que un botón realice una función específica). También es colaborativo y permite la gestión de roles.

En la Figura 4.1 se puede ver la interfaz de un proyecto en blanco. En la parte superior de la página, se encuentran las opciones para crear figuras geométricas y añadir notas, así como agregar a otros usuarios, y exportar el proyecto como una serie de imágenes en formato PNG o PDF y a formato HTML (Figura 4.2). Se observa que en el menú lateral de la izquierda se encuentran los apartados para la creación del prototipo (plantillas, páginas que tiene el proyecto, comentarios, imágenes, iconos, etc.), como se puede ver en la Figura 4.3. Por último, en el menú lateral de la derecha, podemos encontrar el formato de las diferentes páginas (o componentes) (Figura 4.4), y las interacciones disponibles (Figura 4.5).

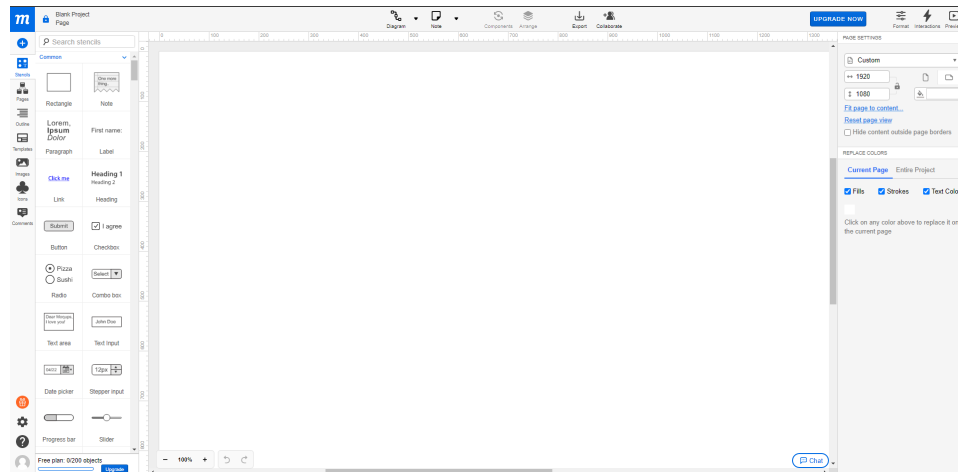


Figura 4.1: Interfaz de Moqups



Figura 4.2: Menú superior

En este proyecto se ha usado Moqups para la creación de la versión 1.0 del prototipo de la aplicación web (en la sección 6.2.2 se puede leer una explicación detallada de cómo se ha creado este prototipo y de las diferentes vistas del mismo).

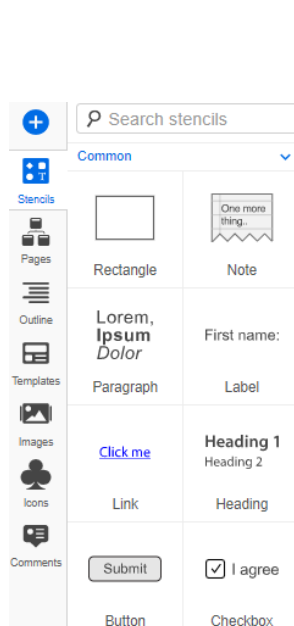


Figura 4.3: Menú lateral izquierda

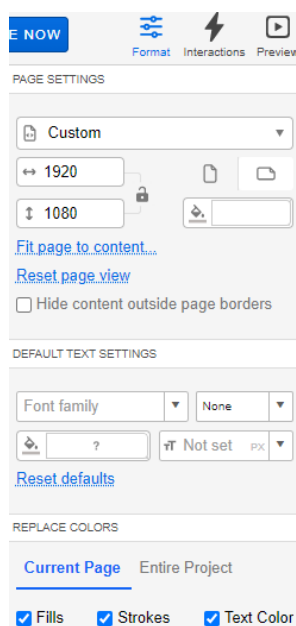


Figura 4.4: Menú lateral derecha (formato)

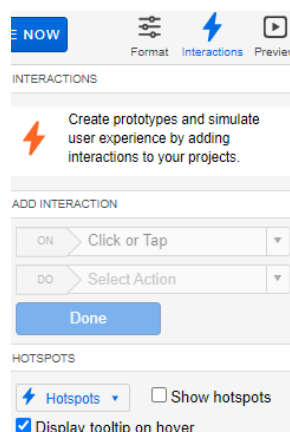


Figura 4.5: Menú lateral derecha (interacciones)

4.2. React

*React*² es una librería de *JavaScript*, creada por Facebook y de código abierto³, que permite crear interfaces de usuario interactivas de forma sencilla. Está basada en la programación orientada a componentes, donde cada componente se puede ver como una funcionalidad distinta, es decir, como una pieza de un puzle. La ventaja de usar componentes es que, al ser independientes unos de otros, si en la carga de una página web falla uno, no afectaría al resto de componentes, por lo que dicha página quedaría cargada sin ese componente. Así mismo, al usar un DOM (Modelo de Objetos del Documento) virtual, deja que la propia librería actualice las partes que han cambiado, en lugar de actualizar todos los componentes.

La sintaxis que emplea *React* es muy parecida a la sintaxis HTML. Para definir los componentes, se emplean etiquetas definidas por el usuario dentro de código *Javascript*. Esta sintaxis se llama *JSX*. No es obligatorio su uso, pero facilita tanto la codificación como la lectura del código. En la Figura 4.6 se puede ver un ejemplo de una funcionalidad sin emplear el formato *JSX* y en la Figura 4.7 la misma funcionalidad, pero usando *JSX*.

²<https://es.reactjs.org/>

³<https://github.com/facebook/react>



```
EDITOR EN VIVO DE JSX ■ JSX?

class HelloMessage extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Hola ",
      this.props.name
    );
  }
}

ReactDOM.render(React.createElement(HelloMessage, { name: "Taylor" })),
document.getElementById('hello-example'));
```

Figura 4.6: Funcionalidad sin usar JSX



```
EDITOR EN VIVO DE JSX ☑ JSX?

class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hola {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('hello-example')
);
```

Figura 4.7: Funcionalidad empleando JSX

React aporta rendimiento, flexibilidad y organización de código, frente a la creación de una página web de forma clásica, es decir, sin usar ninguna librería o *framework*. Tiene una documentación bastante completa⁴, junto con un tutorial para aprender desde cero⁵.

⁴<https://es.reactjs.org/docs/getting-started.html>

⁵<https://es.reactjs.org/tutorial/tutorial.html>

Esta librería la hemos usado en el proyecto para implementar los distintos tipos de adaptaciones en la aplicación web (ver sección 6.1.1).

4.3. API de ARASAAC

El Centro Aragonés para la Comunicación Aumentativa y Alternativa (ARASAAC) proporciona una API pública⁶, cuyo uso es gratuito siempre y cuando la aplicación a desarrollar sea no comercial⁷, debido a que está bajo la licencia de *Creative Commons BY-NC-SA*, es decir, “Reconocimiento-NoComercial-CompartirIgual”⁸ en el que se debe dar crédito a los/as autores/as originales. No se puede usar para fines comerciales y en el caso de que se modifique o transforme ese material se debe distribuir bajo la misma licencia.

La API de ARASAAC consta de cuatro apartados, cada uno de los cuales tiene una ruta web dependiendo de lo que se quiera consultar: materiales, pictogramas, palabras clave y usuarios. Como en este proyecto usaremos solo la búsqueda de pictogramas, nos centraremos en el apartado de pictogramas. La API de ARASAAC proporciona ocho servicios web para pictogramas. Todos ellos se usan con el método de petición *GET* y con formato de salida en *JSON*. De los ocho servicios web proporcionados, en este TFG se usarán los siguientes:

- **/pictograms/{idPictogram}**: Devuelve el pictograma en formato imagen dado su identificador {idPictogram}. Si tiene éxito, devolverá la imagen del pictograma en formato png. En caso contrario, devolverá un mensaje de error.
- **/pictograms/{locale}/{idPictogram}**: Devuelve los datos de un pictograma dado su identificador {idPictogram} e idioma {locale}⁹. Si tiene éxito devolverá una lista en formato *JSON* con los datos de cada pictograma. En caso contrario, devolverá un error 404.
- **/pictograms/{idPictogram}/languages/{languages}**: Devuelve los datos de un pictograma dado su identificador {idPictogram} en uno o más idiomas⁹ especificados en {languages}. Si tiene éxito devolverá

⁶<https://arasaac.org/developers/api>

⁷<https://arasaac.org/developers/>

⁸<https://creativecommons.org/licenses/by-nc-sa/4.0/>

⁹Idiomas disponibles: an, ar, bg, br, ca, de, el, en, es, et, eu, fa, fr, gl, he, hr, hu, it, mk, nl, pl, pt, ro, ru, sk, sq, sv, sr, val, zh

los datos del pictograma en formato *JSON* en los idiomas proporcionados. En caso contrario, devolverá un mensaje de error.

- `/pictograms/{locale}/search/{searchText}`: Devuelve en una lista los datos de los pictogramas en el idioma `{locale}`⁹ que contenga la cadena de caracteres `{searchText}` pasada como parámetro. Si tiene éxito devolverá dicha lista en formato *JSON* con los datos de cada pictograma encontrado. En caso contrario, no devolverá nada.
- `/pictograms/{locale}/bestsearch/{searchText}`: Devuelve en una lista los datos de los pictogramas en el idioma `{locale}`⁹ que contenga **exactamente** la cadena de caracteres `{searchText}` pasada como parámetro. Si tiene éxito devolverá dicha lista en formato *JSON* con los datos de cada pictograma encontrado. En caso contrario, no devolverá nada.

4.4. Jest

*Jest*¹⁰ es una librería para *Javascript*, creada por Facebook, que permite realizar pruebas unitarias y de integración y es compatible con muchos *frameworks* incluyendo el que hemos elegido (*React*). *Jest* tiene una instalación muy sencilla, de pocos pasos, y su configuración es mínima. La documentación¹¹ es completa, y contiene lo necesario para poder desarrollar estos tipos de pruebas, junto con una serie de ejemplos realizados paso a paso. El uso de esta librería proporciona ventajas frente a no usarla o hacer tests de forma tradicional:

- **Compatibilidad.** Es compatible con la mayoría de *frameworks* que existen, tales como Angular, React, NodeJS, Babel, etc.
- **Rapidez.** Cuando las pruebas están vinculadas a la CPU, se ahorra muchísimo tiempo en la ejecución de las pruebas. Esta rapidez se consigue gracias a la combinación de varios factores:
 1. Paralelización. Gracias a la paralelización, se produce una gran ganancia en el rendimiento.
 2. Ejecuta primero las pruebas más lentas. Aprovecha al máximo su capacidad de procesamiento al ejecutar primero las pruebas más lentas.

¹⁰<https://jestjs.io/es-ES/>

¹¹<https://jestjs.io/es-ES/docs/getting-started>

3. Tiene una caché de transformaciones babel incorporada. Aplicar transformaciones al código requiere un uso intensivo de CPU. Gracias al uso de una caché que se comparte entre procesos, se reduce mucho tiempo al ejecutar las pruebas.

- **Instantáneas.** Las pruebas de instantáneas son de gran utilidad para garantizar que la interfaz no cambia de forma inesperada.

Una vez que se ha instalado Jest, para realizar un test hay que seguir estos pasos:

1. Crear un fichero *Javascript* donde estará la función a probar (por ejemplo, *suma.js*):

```
function suma(a,b){  
  return a + b;  
}  
  
module.exports = suma;
```

Listing 4.1: Ejemplo de función a probar, en este caso, suma de dos números

2. Crear un fichero *Javascript* que contendrá la prueba (por ejemplo *suma.test.js*). Para escribir la prueba, tiene que seguir el siguiente esquema:

```
test(título, () =>{  
  expect(valor, función, expresión...).comparador(  
    resultadoEsperado);  
});
```

Listing 4.2: Esquema del test

donde “comparador” puede ser *.toBe()*, *.toEqual()*, *.toBeNull()*, etc., una de las funciones comparadoras que más se ajuste a lo que se quiere probar. En la documentación oficial¹², hay un apartado llamado “Utilizando Comparadores”, donde enseñan algunas funciones comparador. Si se quiere ver todas las funciones que ofrece es mejor visitar la API¹³.

En el caso de la prueba que queremos escribir:

```
const suma = require("./suma");  
  
test("Prueba de suma: 1 + 2 = 3", () =>{  
  expect(suma(1,2)).toBe(3);  
});
```

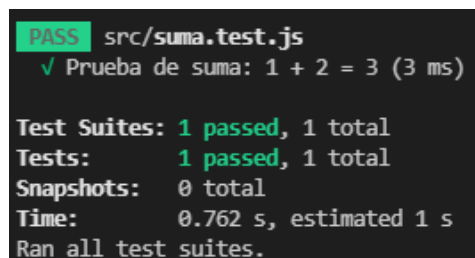
¹²<https://jestjs.io/es-ES/docs/using-matchers>

¹³<https://jestjs.io/es-ES/docs/api>

```
test("Prueba de suma incorrecta: 1 + 2 = 4", () =>{  
  expect(suma(1,2)).toBe(4);  
});
```

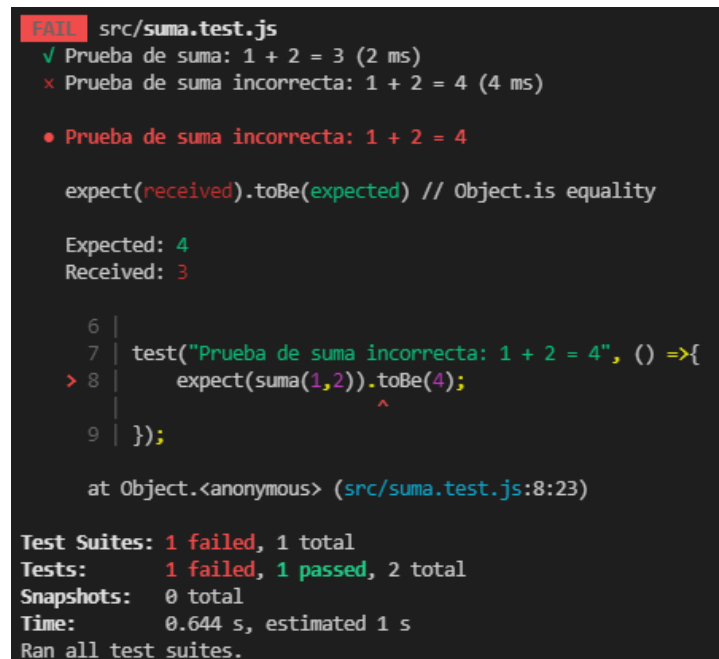
Listing 4.3: Ejemplo de test para la función suma

Si ejecutamos solo la primera función, nos saldrá que hemos pasado el test (Figura 4.8). En cambio, si ejecutamos ambos tests, vemos que el primero sí lo ha pasado, pero el segundo ha fallado, y nos da más información sobre lo que debería haber dado y la línea de código donde ha fallado (Figura 4.9).



```
PASS src/suma.test.js  
✓ Prueba de suma: 1 + 2 = 3 (3 ms)  
  
Test Suites: 1 passed, 1 total  
Tests: 1 passed, 1 total  
Snapshots: 0 total  
Time: 0.762 s, estimated 1 s  
Ran all test suites.
```

Figura 4.8: Test pasado correctamente



```
FAIL src/suma.test.js  
✓ Prueba de suma: 1 + 2 = 3 (2 ms)  
✗ Prueba de suma incorrecta: 1 + 2 = 4 (4 ms)  
  
• Prueba de suma incorrecta: 1 + 2 = 4  
  
expect(received).toBe(expected) // Object.is equality  
  
Expected: 4  
Received: 3  
  
6 |  
7 | test("Prueba de suma incorrecta: 1 + 2 = 4", () =>{  
> 8 |   expect(suma(1,2)).toBe(4);  
   |                     ^  
9 | });  
  
at Object.<anonymous> (src/suma.test.js:8:23)  
  
Test Suites: 1 failed, 1 total  
Tests: 1 failed, 1 passed, 2 total  
Snapshots: 0 total  
Time: 0.644 s, estimated 1 s  
Ran all test suites.
```

Figura 4.9: Test no pasado correctamente

Esta librería la hemos usado en el proyecto para realizar las pruebas unitarias y de integración de las distintas partes de la aplicación.

4.5. CKEditor

*CKEditor*¹⁴ es un editor de código abierto de tipo “WYSIWYG”¹⁵ y es totalmente personalizable. *CKEditor* tiene dos versiones diferentes:

- **CKEditor 4:** Es la versión antigua compatible con la mayoría de los navegadores. Con ella se puede tener tres tipos de editor: *Article Editor* (editor para artículos), *Document Editor* (editor para documentos) e *Inline Editor* (editor en línea). La documentación¹⁶ es muy completa: contiene una guía de instalación paso a paso, además de bastantes ejemplos para diferentes propósitos (insertar contenido, personalizar la interfaz de usuario...). También tiene una demo online¹⁷ que muestra los diferentes editores para esta versión. Gracias al programa LTS (*Long Term Support*) tanto el desarrollo como el soporte de esta versión están garantizados hasta 2023.
- **CKEditor 5:** Es la versión más moderna con un rediseño y mejora de la interfaz y la introducción de un nuevo modelo de datos. A diferencia de la versión anterior, esta versión tiene más tipos de editor: *Classic* (el editor clásico), *Balloon* (editor de globo, que permite insertar contenido directamente en la ubicación que se está señalando), *Balloon Block* (el mismo que el anterior pero se edita por bloques), *Inline* (el mismo que en *CKEditor 4*), *Document* (el mismo que en *CKEditor 4*) y *Pagination* (el editor de paginación, que permite ver las líneas de salto de página que coincidirían con los límites de la página cuando el documento se exporta a PDF o Word). También tiene una demo online¹⁸ de los diferentes tipos de editor para esta versión, y un constructor online¹⁹ donde se puede elegir el tipo de editor y los complementos que tendrá. La documentación²⁰ en esta versión también es bastante completa: tiene una guía de instalación paso a paso, ejemplos de los editores, además de una guía de cómo trabajar con el *framework* de este editor.

¹⁴<https://ckeditor.com/>

¹⁵**WYSIWYG**, acrónimo de *What You See Is What You Get* (en español, “lo que ves es lo que obtienes”). Es una frase aplicada a los procesadores de texto y otros editores con formato, que permiten escribir un documento mostrando directamente el resultado final, frecuentemente el resultado impreso. <https://es.wikipedia.org/wiki/WYSIWYG>

¹⁶<https://ckeditor.com/docs/ckeditor4/latest/>

¹⁷<https://ckeditor.com/ckeditor-4/demo/>

¹⁸<https://ckeditor.com/ckeditor-5/demo/>

¹⁹<https://ckeditor.com/ckeditor-5/online-builder/>

²⁰<https://ckeditor.com/docs/ckeditor5/latest/>

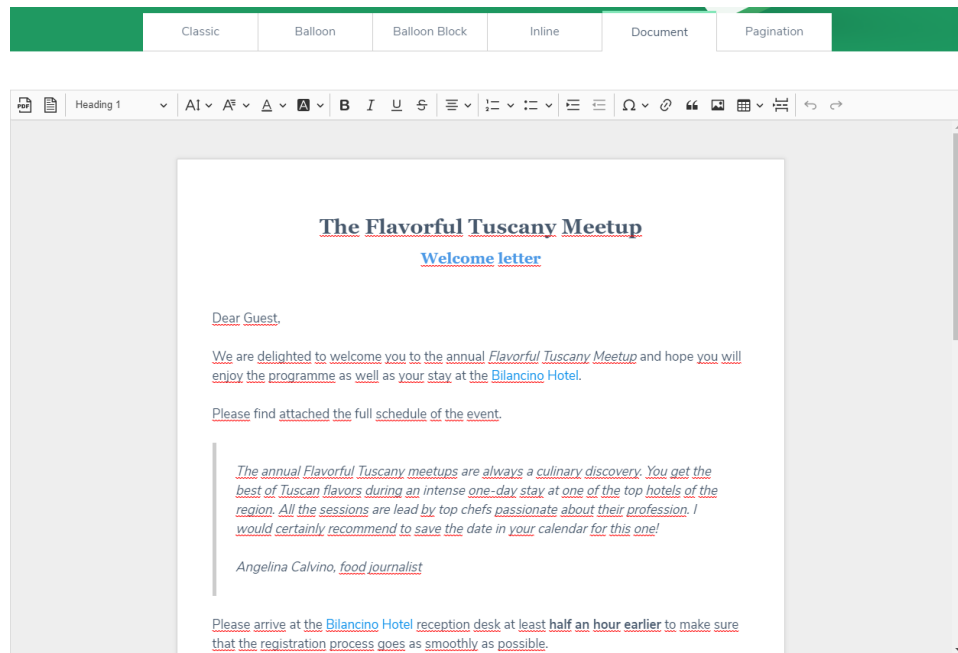


Figura 4.10: Ejemplo de CKEditor 5

En este proyecto hemos usado la versión *CKEditor 5* para crear un documento editable, en el que el usuario puede realizar los diferentes tipos de adaptaciones que ofrece la aplicación, y poder descargar el resultado final en formato PDF. En la Figura 4.10 se puede ver un ejemplo de la versión *CKEditor 5*²¹. En la parte superior se muestra una barra de navegación con los diferentes ejemplos de tipos de editor mencionados anteriormente, y en la inferior, el editor junto con su barra de herramientas (donde el primer elemento de ésta barra es la exportación a formato PDF).

4.6. Word-search

Para la creación de los ejercicios de sopas de letras empleamos el paquete NPM (*Node Package Manager*) *word-search*²². Para crear una sopa de letras con este paquete, se necesitan unas opciones de configuración que están encapsuladas en un objeto, y devuelve la sopa de letras en un array. Las opciones de configuración son las siguientes:

²¹Fuente: <https://ckeditor.com/ckeditor-5/demo/#document>

²²<https://www.npmjs.com/package/@blex41/word-search>

- **cols:** Es un número entero que representa el número de columnas que tendrá la sopa de letras. Por defecto tiene un valor de 10.
- **rows:** Es un número entero que representa el número de filas que tendrá la sopa de letras. Por defecto tiene un valor de 10.
- **disabledDirections:** Es un array de strings que representa las direcciones cardinales ("N", "S", "E", "W", "NE", "NW", "SE", "SW") no válidas para colocar las palabras en la sopa de letras. Por defecto tiene un valor vacío, es decir, todas las direcciones están permitidas.
- **dictionary:** Es un array de strings que contiene las palabras que debe contener la sopa de letras. Por defecto tiene un valor vacío.
- **maxWords:** Es un número entero que representa el número de palabras máximo que se insertarán en la sopa de letras. Por defecto tiene un valor de 20. Aunque la longitud del array *dictionary* supere este valor, solo se insertará ese número de palabras en la sopa de letras.
- **backwardsProbability:** Es un número decimal entre 0 y 1 que representa la probabilidad que tiene cada palabra de escribirse al revés. Solo funciona si no se deshabilita, en una dirección cardinal, uno de los extremos. Por ejemplo, para poder escribir una palabra al revés en vertical, se deben tener las direcciones cardinales "N" y "S" habilitadas, no solo una de ellas. Por defecto tiene un valor de 0.3.
- **upperCase:** Es un booleano que indica si las letras de la sopa de letras se quieren en mayúscula (*true*) o no (*false*). Por defecto tiene el valor *true*.
- **diacritics:** Es un booleano que indica si las letras deberían mantener las tildes en la sopa de letras. Por defecto tiene el valor *false*, es decir, no mantener las tildes en la sopa de letras.

En el caso de que alguna de estas opciones no se modificasen, no daría ningún error, sino que cogería el valor por defecto que tuvieran.

Para crear la sopa de letras, es necesario hacer una llamada a `Wordsearch` pasándole como parámetro las opciones de configuración. En el siguiente ejemplo se ha establecido un tablero de 8x9, se han habilitado todas las direcciones, la probabilidad que tiene cada palabra de escribirse al revés se ha fijado a 0.5 (cada palabra tiene un 50% de probabilidad de escribirse al revés), el número máximo de palabras que se insertarán es 20, se quieren mantener las tildes, las letras se quieren en mayúsculas y hay que buscar las palabras `perro`, `gata`, `flamenco` y `avestruz`.

```
const WordSearch = require("@blex41/word-search");

const options = {
  cols: 9,
  rows: 8,
  disabledDirections: [],
  dictionary: ["perro", "gata", "flamenco", "avestruz"],
  maxWords: 20,
  backwardsProbability: 0.5,
  upperCase: true,
  diacritics: true
};

const ws = new WordSearch(options);
```

Listing 4.4: Generación de la sopa de letras

Una vez creada la sopa de letras, el paquete proporciona las siguientes funciones²³:

- **grid()**: Devuelve un array de caracteres donde cada posición representa una letra de la sopa de letras. En la Figura 4.11 se puede ver un ejemplo del resultado de esta función.



```
Array (8 items)
▶ 0: ["H", "P", "C", "D", "E", "H", "E", "J", "A"]
▶ 1: ["Z", "I", "E", "M", "C", "K", "O", "O", "I"]
▶ 2: ["V", "G", "I", "R", "W", "X", "C", "H", "A"]
▶ 3: ["W", "W", "R", "L", "R", "R", "G", "I", "T"]
▶ 4: ["S", "K", "H", "U", "D", "O", "B", "E", "A"]
▶ 5: ["A", "V", "E", "S", "T", "R", "U", "Z", "G"]
▶ 6: ["C", "F", "L", "A", "M", "E", "N", "C", "O"]
▶ 7: ["R", "N", "T", "J", "R", "W", "L", "O", "J"]
▶ Array Prototype
```

Figura 4.11: Ejemplo de resultado de sopa de letras usando grid()

- **toString()**: Devuelve un array de strings donde cada posición representa una fila de la sopa de letras. En la Figura 4.12 se puede ver un ejemplo del resultado de esta función.
- **words()**: Devuelve un array de objetos con las palabras que se han

²³Para cada ejemplo, se han usado las opciones de configuración del Listing 4.4. Así mismo, se han ejecutado dichas funciones una sola vez y seguidas, dando como resultado el mismo tablero.

```

H P C D E H E J A
Z I E M C K O O I
V G I R W X C H A
W W R L R R G I T
S K H U D O B E A
A V E S T R U Z G
C F L A M E N C O
R N T J R W L O J

```

Figura 4.12: Ejemplo de resultado de sopa de letras usando toString()

insertado correctamente en la sopa de letras. Cada objeto tiene tres campos:

- **word**: la palabra que está en el campo *dictionary*.
- **clean**: este campo depende de las opciones de *upperCase* y *diacritics*, es decir, se escribirá la palabra del campo *word* con o sin mayúsculas y con o sin tildes (dependiendo si los valores de dichas opciones están a *true* o *false*).
- **path**: contiene un array de objetos con las posiciones “x” e “y” de cada letra de la palabra en la sopa de letras.

En la Figura 4.13 se puede ver un ejemplo del resultado cuando se usa la función words().

```

Array (4 items)
  0: Object
    word: "avestruz"
    clean: "AVESTRUZ"
    path: [Object {x: 0, y: 5}, Object {x: 1, y: 5}, Object {x: 2, y: 5}, Object {x: 3, y: 5}, Object {x: 4, y: 5}, ...]
  1: Object {word: "flamenco", clean: "FLAMENCO", ...}
  2: Object {word: "gata", clean: "GATA", ...}
  3: Object {word: "perro", clean: "PERRO", ...}
  Array Prototype

```

Figura 4.13: Ejemplo de las palabras insertadas al usar words()

- **dump()**: Devuelve un objeto que representa el estado de la sopa de letras (es una *snapshot* del tablero). Esto puede resultar útil si se quiere guardar el juego y usarlo después con la siguiente función. En la Figura

4.14 se puede ver un ejemplo del objeto resultante cuando se usa esta función.



Figura 4.14: Ejemplo del objeto resultante al usar `dump()`

- **load(backup)**: Carga el parámetro *backup* que representa un estado de la sopa de letras generado previamente con *dump()*.
- **read(start, end)**: Devuelve un string con las letras que se forman al leer las posiciones de *start* a *end*. En la Figura 4.15 se puede ver la forma en la que se llama a esta función, pasándole los objetos posición *start* y *end* y, en la Figura 4.16, un ejemplo del string formado al leer dichas posiciones.

```
ws.read({ x: 1, y: 0 }, { x: 5, y: 4 })
```

Figura 4.15: Ejemplo de llamada a la función `read()`

"PERRO"

Figura 4.16: Ejemplo del string formado cuando se usan los parámetros de la Figura 4.15

Capítulo 5

Metodología de desarrollo

RESUMEN: En este capítulo se describe la metodología de desarrollo que se usa en el proyecto. En la sección 5.1 se hace una introducción de los tipos de metodologías que existen. En la sección 5.2 se explica la metodología que se ha elegido. Por último, en la sección 5.3 se describen los tipos de pruebas que se realizarán a lo largo del proyecto.

5.1. Introducción

La metodología (del griego *metá* “más allá”, *odós* “camino” y *logos* “razón, estudio”) hace referencia al “conjunto de métodos que se siguen en una investigación científica o una exposición doctrinal”¹. Si este término se lleva al ámbito de la gestión de proyectos, se puede definir como un conjunto de técnicas, herramientas y procedimientos que permite organizar los procesos de un proyecto.

Existen dos tipos de metodologías principalmente: tradicionales y ágiles. Las metodologías tradicionales “buscan imponer disciplina al proceso de desarrollo de software y de esa forma volverlo predecible y eficiente” (Cadavid, Martínez y Vélez, 2013), y están orientadas a procesos. El desarrollo del proyecto se inicia mediante una serie de etapas: captura de requisitos, análisis, diseño y desarrollo. En la primera etapa, captura de requisitos, existe una abundante comunicación con el cliente, al contrario que en la etapas posteriores, en las que apenas existe comunicación (prácticamente nula), debido a que los requisitos se deben quedar fijos desde el principio, por lo que no

¹<https://dle.rae.es/metodología>

se esperan cambios en ellos a lo largo del proyecto. Así mismo, la entrega de software se realiza al finalizar el desarrollo, por lo que el cliente solo puede ver el resultado al final. Este tipo de metodología se suele usar cuando hay un problema conocido y se sabe su solución o si los requisitos están muy claros desde el principio. Algunos ejemplos de metodologías tradicionales son: *ITIL*², *Métrica 3*³, *RUP*⁴, etc.

Por otro lado, las metodologías ágiles tienen un enfoque adaptativo, en el que cualquier cambio se acoge con normalidad (se espera que ocurran cambios). Al contrario que las metodologías tradicionales, las cuales estaban orientadas a los procesos, las metodologías ágiles están orientadas a las personas. En estas metodologías lo que se pretende es generar valor para el cliente, por lo que se necesita un representante de éste, que puede ser un miembro más del equipo, y, sobre todo, una comunicación constante con él. Uno de los objetivos de las metodologías ágiles es conseguir, lo antes posible, un producto que sea funcional y que genere valor al cliente. Esto se logra a través de constantes entregas de software, lo que implica que el cliente pueda proporcionar *feedback* que permita aumentar dicho valor. Este tipo de metodología funciona bien cuando los requisitos no están claros y tampoco lo está el problema a solucionar y la manera en la que se puede desarrollar. Algunos ejemplos⁵ de metodologías ágiles son: *Scrum*, *Extreme Programming (XP)*, *Kanban*, etc.

Este proyecto seguirá la metodología *Kanban* debido a que es menos prescriptivo que otras metodologías ágiles (tiene solo tres reglas) y da más flexibilidad a la hora de organizar y desarrollar el trabajo y de adaptarse a los cambios que puedan surgir durante el proyecto. En la siguiente sección se cuenta con más detalle la metodología *Kanban*.

5.2. Kanban

El término “*Kanban*”⁶ proviene del japonés, cuyo significado es “tarjetas visuales”. Fue creado en la empresa Toyota en la década de los 50 para controlar el avance del trabajo con los materiales disponibles.

Kanban es menos prescriptivo que otras metodologías ágiles como *Scrum*

²<https://www.heflo.com/es/blog/itil/que-es-metodologia-itil/>

³https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html

⁴<http://ima.udg.edu/~sellares/EINF-ES2/Present1011/MetodoPesadesRUP.pdf>

⁵<https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/>

⁶<https://blog.trello.com/es/metodologia-kanban>

y tiene tres reglas principales (Garzas, 2011):

1. **Visualizar el trabajo y las fases del ciclo de producción o flujo de trabajo.** El trabajo está dividido en partes o tareas. Para visualizar todas estas tareas, se usa una pizarra o tablero llamado “tablero *Kanban*”. Una de las ventajas de usar el tablero es que cualquier persona del equipo puede saber el trabajo realizado y el trabajo pendiente, evitando que éste último se acumule. Además, permite conocer en qué tarea está trabajando cada uno. El tablero *Kanban* se divide en varias columnas que representan las distintas fases por las que puede pasar una tarea. Las fases las decide cada equipo, aunque el modelo más común suele tener las siguientes:
 - **To Do:** Tareas pendientes.
 - **Doing:** Tareas en curso.
 - **Done:** Tareas terminadas, validadas y aprobadas. Una vez en esta fase, no se podrá volver a mover a ninguna anterior, ya que el cliente lo ha validado, y le ha generado valor.
2. **Determinar y respetar el trabajo en curso.** Debe haber un límite máximo de tareas que se pueda realizar en cada fase para evitar cuellos de botella. Este límite, llamado *Work In Progress* (WIP), debe ser algo conocido y hay varias formas de calcularlo. Lo suele establecer el equipo de desarrollo y se puede cambiar en cualquier momento. El WIP impide comenzar tareas hasta que no se hayan finalizado otras en curso, evitando así la acumulación de tareas.
3. **Medir el tiempo en completar una tarea.** Se pueden distinguir dos tipos de tiempo:
 - **Lead Time** o tiempo de entrega: es el tiempo en el que se tarda en completar una tarea, desde que entra en el flujo de trabajo hasta que se realiza su entrega. Este tiempo hay que medirlo siempre.
 - **Cycle Time** o tiempo de ciclo: es el tiempo que pasa el equipo trabajando en una tarea, desde que se inicia su desarrollo hasta que se da por terminada.

Gracias a estos tiempos podemos saber la productividad y eficiencia del equipo, y ajustar, en caso necesario, el flujo de trabajo.

En nuestro proyecto, distinguimos dos tipos de tareas:

- **Tareas de memoria:** Son las tareas de redacción de la memoria.

- **Tareas de implementación:** Son tareas de diseño y/o desarrollo de código.

En la Figura 5.1 se puede ver nuestro tablero *Kanban*. Este tablero tiene un total de seis columnas:

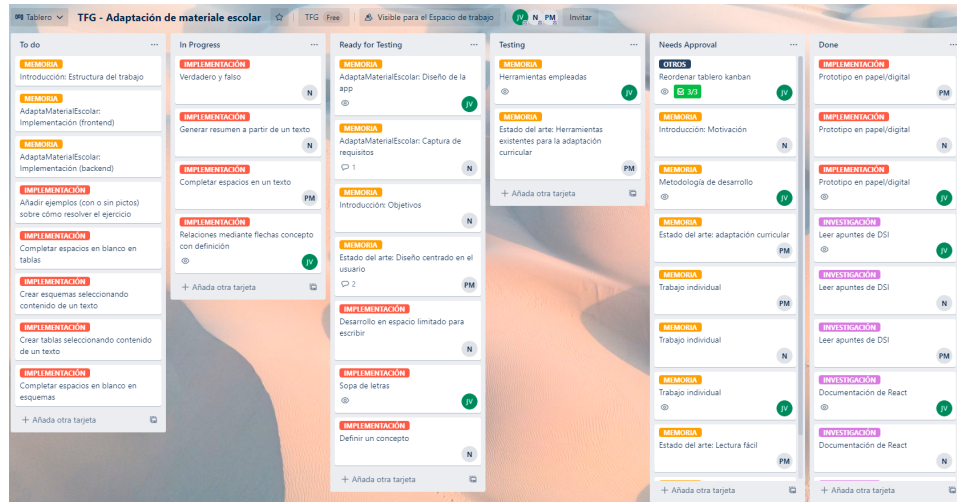






Figura 5.1: Tablero *Kanban*

- **To Do:** Se corresponden con las tareas que todavía no se han realizado.
- **In Progress:** Son tareas en las que el equipo de desarrollo ha comenzado a trabajar.
- **Ready for Testing:** Son tareas listas para ser probadas.
- **Testing:** Son tareas que se están probando. Dependiendo del tipo de tarea se probará de una forma u otra. En la sección 5.3 se dan más detalles de cómo se prueban las tareas de memoria y las de implementación.
- **Needs approval:** En esta lista se encuentran las tareas que necesitan ser aprobadas por las tutoras antes de darlas por finalizadas. Una vez que se ha terminado de trabajar en una tarea y se ha acabado de hacerle las pruebas, habrá que moverla de *Testing* a esta columna.
- **Done:** Son tareas que se han dado por terminadas, validadas y aprobadas. Cuando las tutoras le han dado el visto bueno, se podrá mover de *Needs Approval* a *Done*.

Así mismo, en la Figura 5.1 se observa que las tareas tienen asignadas unas etiquetas de colores:

-  Tareas relacionadas con la memoria.
-  Tareas correspondientes con diseño y desarrollo de código (implementación).
-  Tareas que requieren investigación por parte del equipo, antes de empezar a codificar o realizar cualquier otro tipo de tarea.
-  Tareas que no corresponden con ninguna de las anteriores.

El WIP será de dos tareas por persona en cada columna (*In Progress* y *Testing*), lo que hace un total de seis tareas en cada columna.

5.3. Tipos de pruebas

Como tenemos dos tipos de tareas, de memoria y de implementación, cada una se tratará de una forma diferente a la hora de realizar las pruebas.

5.3.1. Pruebas de memoria

Las pruebas de memoria consisten en buscar errores léxicos y gramaticales en la memoria, y corregirlos antes de su entrega, además de completarla con más información en el caso de que sea posible, y comprobar que todo se entiende. Estas pruebas las harán todos los miembros del equipo de la siguiente forma: cuando haya alguna tarea de memoria en *Ready for testing*, el usuario que vaya a revisarla, se la asignará y la llevará a *Testing*. Una vez que haya terminado, y no sea la última persona en haberla revisado, la volverá a llevar a *Ready for testing*. En caso contrario, la llevará a *Needs approval*. Con este sistema tratamos de asegurarnos de que todo esté correctamente escrito.

5.3.2. Pruebas de implementación

Para las pruebas de implementación, se contemplan dos tipos de pruebas: pruebas unitarias y pruebas de integración. Todas estas pruebas las haremos

con *Jest* (ver sección 4.4). Estas pruebas las desarrollará y realizará alguno de los miembros que no haya implementado esa parte del código, y las hará cuando la tarea correspondiente de prueba esté en la columna *Ready for Testing*. La razón de esto es porque las personas que no han escrito el código pueden sacar más casos de prueba que las personas que lo han escrito.

5.3.2.1. Pruebas unitarias

Una prueba unitaria se utiliza para comprobar que un método implementado funciona como se esperaba. Debe cumplir una serie de características:

- Deben ser **automáticas**: se deben poder ejecutar sin que haya una intervención manual.
- Deben ser **completas**: es decir, deben cubrir la totalidad del código.
- Deben ser **independientes**: debido a que se han creado para comprobar una parte concreta del código, no deberían interferir con otras partes, y se deben poder ejecutar en cualquier entorno.
- Deben ser **repetibles**: se deben repetir todas las veces que queramos, y el resultado debe ser el mismo en todas las repeticiones.

El uso de pruebas unitarias proporciona una serie de ventajas:

- **Aumento de la calidad** del código: debido a que estas pruebas se ejecutan de forma regular, permiten detectar errores a tiempo y poder corregirlos antes de completar el código y liberar la aplicación.
- **Facilitan los cambios**: si se considera que el código es mejorable se pueden aplicar cambios sin problemas, ya que la prueba unitaria nos avisaría de los errores que pudieran surgir.
- **Reduce los tiempos** de integración, ya que podemos probar partes del código sin disponer del código completo.
- **Reduce el coste**: teniendo en cuenta que permite detectar errores en fases tempranas, evitan el paso de los errores a fases posteriores y, además, el cambio de corregirlos en fases tempranas es menos costoso y más fácil que si se corrigieran en fases más tardías, ya que en este último caso el error suele ser producido a causa de muchos cambios. Por ello, las pruebas unitarias ayudan a reducir los costes de corrección de errores y a optimizar las entregas, ahorrando tiempo y recursos.

5.3.2.2. Pruebas de integración

Una prueba de integración se utiliza para comprobar que dos o más componentes, ya validados por las pruebas unitarias, son compatibles entre sí y funcionan correctamente. Hay varios tipos de pruebas de integración (Rodríguez, 2015):

- **Top-Down:** Es una estrategia de “arriba a abajo”, donde se va probando los componentes que no son llamados por ningún otro, y se integran los componentes que son llamados desde la parte integrada.
- **Bottom-Up:** Es una estrategia de “abajo a arriba”, donde se prueban los componentes que no llaman a otras partes de la aplicación, y se continua por componentes que solo llaman a la parte integrada.
- **End-to-End:** Es una estrategia orientada al proceso de negocio, en la cual integran los componentes necesarios por parte de un proceso de negocio para ver si el flujo de la aplicación funciona tal y como fue diseñado.
- **Funciones:** Este tipo de integración está orientada a una función del sistema. Se van integrando cada uno de los componentes que necesita esa función.
- **Ad-Hoc:** En esta estrategia el software se ve como módulos independientes, y una vez que se han implementado y validado por parte del equipo de desarrollo, se integran.
- **Big-Bang:** Esta estrategia solo es válida cuando se dispone de todos los componentes, es decir, no se integra hasta que el software no esté completo.

Este proyecto seguirá la estrategia *Bottom-up*.

Capítulo 6

AdaptaMaterialEscolar

RESUMEN: En este capítulo se explicará, en la sección 6.1, para quién va a desarrollarse el proyecto de AdaptaMaterialEscolar y el proceso de captura de requisitos. En la sección 6.2, se explicará cómo se diseñó la aplicación a partir de los requisitos. Por último, en la sección 6.3, se mostrará cómo se ha implementado la aplicación.

6.1. Requisitos de la aplicación

El Trabajo de Fin de Grado de AdaptaMaterialEscolar es un proyecto sin ánimo de lucro que ha sido desarrollado con la colaboración de las profesoras del Aula TEA del IES Maestro Juan de Ávila de Ciudad Real.

Para su implementación, debido a que se pretende que el proyecto sea realmente útil para los docentes, se hizo una amplia captura de requisitos con los usuarios finales, de forma que se pudieron conocer las necesidades reales de un centro educativo, siguiendo un Diseño Centrado en el Usuario.

Se realizaron una serie de reuniones con los usuarios finales, tanto presenciales como online, para obtener los requisitos de la aplicación; además, se mantuvo el contacto con ellos durante todo el proceso de diseño e implementación, para irles enseñando la evolución de la aplicación.

6.1.1. Captura de requisitos

El 25 de julio de 2019 se organizó una primera reunión presencial con las dos profesoras que forman parte del Aula TEA en el IES Maestro Juan de Ávila: Ana María Alonso Frades, especialista en Pedagogía Terapéutica y Ana María Díaz Valle, especialista en Audición y Lenguaje.

Las profesoras nos hablaron sobre los diferentes perfiles de alumnos TEA con los que trabajaban en el centro y explicaron la importancia de contar con una aplicación especializada en adaptaciones curriculares, que fuera flexible y permitiera generar diferentes modelos de temario, actividades y exámenes con el menor esfuerzo posible. El proyecto debía poder convertir un mismo párrafo o ejercicio en otro más sencillo de comprender, asimilar o resolver, de forma que pudiera personalizarse para cada alumno de acuerdo a su capacidad cognitiva y así facilitarles el aprendizaje sin excluir temario.

A pesar de que hay unas pautas a seguir para adaptar el temario, las actividades y los exámenes (por ejemplo usar letra grande e imágenes), no hay un estándar definido que sigan todos los profesores, y a algunos alumnos les resulta confuso el cambio de una asignatura a otra.

Después de hablar con las profesoras y conocer a un alumno TEA que nos explicó su rutina en el instituto, vimos un ejemplo de tema y examen adaptado de Ciencias Naturales.

En este punto, a partir de la información proporcionada por los usuarios finales, se tomaron las siguientes decisiones:

- Tipo de aplicación: Inicialmente las profesoras plantearon desarrollar una aplicación de escritorio que pudieran instalar tanto en los ordenadores del centro, como en los suyos propios, pero se tuvo en cuenta que probablemente no todo el profesorado contaría con un ordenador personal con el que poder trabajar desde casa. Por ello, se tomó la decisión de desarrollar el proyecto como aplicación web, de manera que se pudiera utilizar desde cualquier dispositivo con acceso a Internet.
- Registro: Las profesoras indicaron que era mejor no tener que registrarse en la aplicación para poder trabajar con ella, ya que se consideró que no era conveniente almacenar usuarios ni contraseñas para evitar que los profesores tuvieran que utilizar datos personales con los que identificarse.

A partir de ahí, como resultado de la reunión, obtuvimos las funcionali-

dades de la aplicación. A continuación, presentamos estas funcionalidades:

- Adaptaciones de temario: La motivación de las adaptaciones de temario es conseguir, en el menor tiempo posible, una unidad didáctica teórica adaptada al alumno de forma casi inmediata y automática, seleccionando el texto a convertir. En este sentido, la aplicación debe ofrecer al usuario la posibilidad de:
 - Generar un resumen a partir de un texto.
 - Crear esquemas que faciliten la visualización del temario y/o las actividades, seleccionando contenido de un texto ya redactado o escribiendo desde cero.
 - Crear tablas que organicen el temario y/o las actividades, seleccionando contenido de un texto ya redactado o escribiéndolo desde cero.
 - Creación de ejercicios: La motivación de la creación de ejercicios es crear ejercicios de forma fácil, seleccionando texto o un ejercicio ya redactado y adaptándolo de forma automática al tipo de ejercicio que se quiere obtener. La aplicación debe permitir crear los siguientes tipos de ejercicios:
 - Relacionar contenido mediante flechas.
 - Sopa de letras.
 - Completar espacios en blanco en un texto, tabla o esquema.
 - Pregunta de desarrollo con un espacio limitado para escribir la respuesta.
 - Verdadero o falso.
- Además, todos los ejercicios que se creen deben permitir añadir un enunciado y ejemplos (con o sin pictogramas) explicando cómo debe resolverse el ejercicio.
- Adaptaciones de formato: Las adaptaciones de formato deben permitir estandarizar cualquier documento que se vaya a entregar al alumno, de manera que el cambio de una asignatura a otra no suponga un cambio visual y el aprendizaje sea menos costoso. Nuestra aplicación debe permitir al usuario final que, dado un texto, pueda:
 - Sustituir una palabra por un pictograma.
 - Sustituir una palabra por una imagen.
 - Resaltar una palabra con un color, para que los alumnos visualicen cada una de las palabras resaltadas en el texto con mayor facilidad.

- Añadir una leyenda de colores con la categoría de cada tipo de palabra resaltada, para que los alumnos puedan identificar el significado de cada una de las palabras resaltadas en el texto.
- Añadir una leyenda de colores para el tema de cada asignatura, para que los alumnos las distingan con mayor facilidad.
- Estandarizar el formato en textos, estableciendo una fuente de texto determinada y un tamaño acorde para cada alumno.
- Estandarizar el formato para títulos e índices del temario, generando de forma automática una plantilla del color de la asignatura.
- Aumentar el tamaño de la fuente de una parte del texto.
- Subrayar una parte del texto.
- Poner en negrita una parte del texto.
- Añadir imágenes buscando una palabra en bases de datos de imágenes libres.

6.1.2. Análisis de requisitos

Una vez recopilados los requisitos del proyecto, redactamos una lista con éstos para que pudiéramos, tanto las profesoras del Aula TEA como el equipo de desarrollo, analizarlos de forma independiente, para no influir en la toma de decisiones sobre la importancia y dificultad de cada funcionalidad.

Enviamos el listado de requisitos a las profesoras por correo electrónico, con la finalidad de que revisaran el contenido o añadieran nuevas funcionalidades si fuera necesario y evaluaran cada uno de ellos. Las profesoras Ana María Alonso Frades y Ana María Díaz Valle debían puntuar los requisitos según la utilidad que pudieran aportar cada uno de ellos a su trabajo. Las posibles puntuaciones debían ir de uno a tres, siendo el uno poco importante, el dos importante y el tres imprescindible. Prácticamente todos los requisitos fueron puntuados como imprescindibles, por lo que quedó claro que el proyecto cubre una necesidad real y no añadieron ningún requisito más.

Por otro lado, cada miembro del equipo de desarrollo puntuó los requisitos de manera individual, para no influir en la decisión de los compañeros, según la dificultad que consideraban que tendría la implementación de cada requisito a nivel técnico (en el Anexo A.1 se encuentra la tabla con las puntuaciones individuales para cada requisito de cada miembro del equipo de desarrollo). Las posibles puntuaciones debían ir de uno a tres, siendo el uno difícil, el dos intermedio y el tres fácil.

La finalidad de este análisis de los requisitos era obtener un listado con las funcionalidades ordenadas por dificultad e importancia. Con la puntuación que asignaron las profesoras de forma conjunta, se obtuvo el indicador de importancia de cada tarea y, con la media de las puntuaciones del equipo de desarrollo, el indicador de dificultad. En la Tabla 6.1 se muestra el listado final de las funcionalidades, agrupadas por tipo de adaptación, con la media de las puntuaciones obtenidas en importancia y dificultad.

ADAPTACIONES DE TEMARIO (T)		
Requisito	Evaluación	
	Imp.	Dif.
Generar un resumen a partir de un texto	3	2
Crear esquemas que faciliten la visualización del temario y/o las actividades, seleccionando contenido de un texto ya redactado o escribiendo desde cero	3	1.7
Crear tablas que organicen el temario y/o las actividades, seleccionando contenido de un texto ya redactado o escribiendo desde cero	2	1.7

ADAPTACIONES DE EJERCICIOS (E)		
Requisito	Evaluación	
	Imp.	Dif.
Ejercicios de relacionar contenido mediante flechas	3	2
Ejercicios de sopa de letras	2	3
Ejercicios de completar espacios en un texto	3	2.7
Ejercicios de pregunta de desarrollo con un espacio limitado para escribir la respuesta	3	2.7
Ejercicios de verdadero o falso	3	3
Ejercicios de completar los espacios en blanco en tablas	3	2
Ejercicios de completar los espacios en blanco en esquemas	3	1
Añadir enunciados y ejemplos (con o sin pictogramas), donde se explique cómo debe resolverse el ejercicio	2	3

ADAPTACIONES DE FORMATO (F)		
Requisito	Evaluación	
	Imp.	Dif.
Sustituir una palabra por un pictograma	2	2.7
Sustituir una palabra por una imagen	2	2.3
Resaltar una palabra con un color	3	3
Añadir una leyenda de colores con la categoría de cada tipo de palabra resaltada, para que los alumnos puedan identificar el significado de cada una de las palabras resaltadas en el texto	1	3

Tabla 6.1 continuada de la página anterior

Añadir una leyenda de colores para el tema de cada asignatura, para que los alumnos las distingan con mayor facilidad	1	3
Estandarizar formato en textos, estableciendo una fuente de texto determinada y un tamaño acorde para cada alumno	3	2
Estandarizar formato para títulos e índices del temario, generando de forma automática una plantilla del color de la asignatura	3	2
Aumentar el tamaño de la fuente de una parte del texto	3	2.7
Subrayar una parte del texto	2	2.7
Poner en negrita una parte del texto	3	2.7
Añadir imágenes buscando una palabra en bases de datos de imágenes libres	3	2.3
Añadir espacio extra entre líneas para escribir	3	2

Tabla 6.1: Puntuación por importancia (Imp.) y dificultad (Dif.) de todos los requisitos de la aplicación

A continuación, para cada requisito, se multiplicó la importancia por la dificultad para obtener un valor que permitiese ordenar los requisitos por prioridad para su implementación. Para establecer un orden en el desarrollo de los requisitos, se considera más prioritaria la implementación de las funcionalidades con puntuación 9 (imprescindibles y fáciles de implementar) y las menos prioritarias serán aquellas con puntuación 1 (funcionalidades poco importantes y difíciles de implementar). La Tabla 6.2 contiene las funcionalidades de la aplicación: las adaptaciones de temario (T) y ejercicios (E), y la Tabla 6.3 contiene las adaptaciones de formato (F) que serían necesarias en el proyecto. En ambas tablas los requisitos están ordenados de forma descendente por la puntuación final (importancia * dificultad).

Tipo	Requisito	Puntuación
E	Ejercicios de verdadero o falso	9
E	Ejercicios de completar espacios en un texto	8.1
E	Ejercicios de pregunta de desarrollo con un espacio limitado para escribir la respuesta	8.1
T	Generar un resumen a partir de un texto	6
E	Ejercicios de relacionar contenido mediante flechas	6
E	Ejercicios de sopas de letras	6
E	Añadir ejemplos (con o sin pictogramas) sobre cómo resolver el ejercicio	6

Tabla 6.2 continuada de la página anterior

E	Ejercicios de completar los espacios en blanco en tablas	6
T	Crear esquemas que faciliten la visualización del temario y/o las actividades, seleccionando contenido de un texto ya redactado o escribiendo desde cero	5.1
T	Crear tablas que organicen el temario y/o las actividades, seleccionando contenido de un texto ya redactado o escribiendo desde cero	3.4
E	Ejercicios de completar los espacios en blanco en esquemas	3

Tabla 6.2: Lista de funcionalidades relacionadas con adaptaciones de temario (T) y ejercicios (E), ordenada por prioridad

Tipo	Requisito	Puntuación
F	Resaltar una palabra con un color	9
F	Aumentar el tamaño de la fuente de una parte del texto	8.1
F	Poner en negrita una parte del texto	8.1
F	Añadir imágenes buscando una palabra en bases de datos de imágenes libres	6.9
F	Añadir enunciados y ejemplos (con o sin pictogramas), donde se explique cómo debe resolverse el ejercicio	6
F	Estandarizar formato para títulos e índices del temario, generando de forma automática una plantilla del color de la asignatura	6
F	Estandarizar formato en textos, estableciendo una fuente de texto determinada y un tamaño acorde para cada alumno	6
F	Añadir espacio extra entre líneas para escribir	6
F	Sustituir una palabra por un pictograma	5.4
F	Subrayar una parte del texto	5.4
F	Añadir una leyenda de colores con la categoría de cada tipo de palabra resaltada, para que los alumnos puedan identificar el significado de cada una de las palabras resaltadas en el texto	3

Tabla 6.3: Lista de funcionalidades de adaptación de formato, ordenada por prioridad

6.1.3. Requisitos adicionales

El miércoles 4 de diciembre de 2019, el grupo de investigación NIL (Natural Interaction based on Language) organizó un workshop en la Facultad de Informática de la Universidad Complutense con docentes de otros centros y asociaciones. Durante este workshop se mostraron las herramientas tecnológicas inclusivas que se habían desarrollado o que se encontraban en desarrollo en el grupo, siendo una de las presentadas AdaptaMaterialEscolar.

Un miembro del equipo de desarrollo, Jorge Velasco Conde, realizó una presentación en la que explicó a los asistentes el funcionamiento y la finalidad de nuestro proyecto, con ejemplos visuales sobre posibles adaptaciones que podrían llegar a realizarse. Después de la presentación se realizó una ronda de opiniones con todos los docentes y expertos, para que pudieran expresar qué les había parecido la propuesta. Todos los comentarios fueron muy positivos y preguntaron cuándo estaría disponible para uso público. Además de dar un feedback positivo sobre el proyecto, también propusieron un nuevo tipo de adaptación de formato, para mejorar las opciones de personalización para los alumnos. El requisito consiste en aportar un espacio lo suficientemente grande, ampliando la distancia entre los renglones si es necesario, para que los alumnos que tienen una letra de mayor tamaño puedan escribir sin problema.

En este caso únicamente realizamos la evaluación del requisito los desarrolladores, ya que, al ser una petición, se consideró que el indicador de importancia debía tener la máxima puntuación, para poder aportar utilidad a un mayor número de centros y asociaciones y el indicador de dificultad volvió a calcularse de forma independiente entre el equipo, haciendo la media entre las puntuaciones. El requisito se encuentra anexado en la Tabla 6.1 y en la Tabla 6.3 se puede ver su puntuación final (importancia * dificultad).

6.2. Diseño de la aplicación

El diseño de la aplicación web se ha desarrollado en varias iteraciones. En la primera iteración se creó un boceto de la aplicación en papel. En la segunda iteración se mejoró ese boceto, creando un prototipo por vistas a través de la herramienta online *Moqups*. En la tercera iteración cada uno de los miembros del equipo creó su propio prototipo, sin influir los unos con los otros, para ver si el prototipo creado con *Moqups* podía ser mejorado. En la última iteración se realizó el diseño final de la aplicación. En las siguientes secciones se dan más detalles sobre estas iteraciones.

6.2.1. Primera iteración: boceto inicial

Para comenzar el proceso de diseño se creó un boceto en papel de la página principal de la aplicación (Figura 6.1). En la parte superior de este diseño se observa que hay una barra de navegación con dos menús: Texto y Actividades. Cada uno de ellos tiene un submenú con las diferentes adaptaciones disponibles para dicho menú. En el lado derecho encontramos otro menú con dos zonas diferenciadas: en la parte superior tenemos los botones de Subir texto (que sube un fichero en formato *.pdf* o *.docx*) y Descargar (que descarga el documento adaptado en formato *.pdf*). En la parte inferior del menú lateral, nos encontramos un Resaltar información, el cual sirve para dar formato a los títulos, al cuerpo, etc. Por último, en el centro de la página, se encuentra el fichero subido con el que podremos seleccionar textos y poder adaptarlos conforme así se requiera.



Figura 6.1: Boceto inicial de la aplicación

6.2.2. Segunda iteración: versión 1.0 del prototipo

Partiendo del diseño inicial, y a través de la herramienta *Moqups* (ver sección 4.1), se creó un prototipo más amplio por vistas:

- **Página principal:** En la Figura 6.2 se puede ver la página principal de la aplicación. Ésta es la página que se vería nada más entrar. En la parte superior, encontramos dos barras de navegación con diferentes

menús. En la barra de navegación superior, podemos ver los menús relacionados con un editor de texto: Archivo, Edición, Formato, Insertar y Ayuda. En la barra de navegación inferior se encuentran los menús correspondientes a las adaptaciones de Temario y Actividades, además de la posibilidad de buscar imágenes libres¹ (también llamadas imágenes *CC0* o *Creative Commons Zero*²) y pictogramas. En el centro de la página web podemos observar dos elementos: el logo de la aplicación AdaptaMaterialEscolar y una zona para poder subir el fichero en formato *.pdf* o *.docx*, ya sea seleccionándolo de una carpeta o arrastrando desde la carpeta a la zona de subida.



Figura 6.2: Versión 1.0 de la página principal

- **Editor:** Una vez subido el fichero se pasa a la vista del editor. En la Figura 6.3 se puede observar dicha vista. En ésta, encontramos en la parte superior las dos barras de navegación descritas en el punto anterior. En el centro de la página observamos dos elementos: el elemento de la izquierda es el fichero subido, donde se podrán seleccionar las partes del texto que se quieran adaptar. El elemento de la derecha es el editor donde se podrá, también, adaptar e insertar imágenes, pictogramas, dar formato, etc.
- **Búsqueda de pictogramas:** En la Figura 6.4 se puede ver la búsqueda de pictogramas. En la parte superior derecha de la aplicación

¹Las imágenes libres son aquellas imágenes que son de dominio público y que no tienen derechos reservados.

²<https://creativecommons.org/publicdomain/zero/1.0/deed.es>

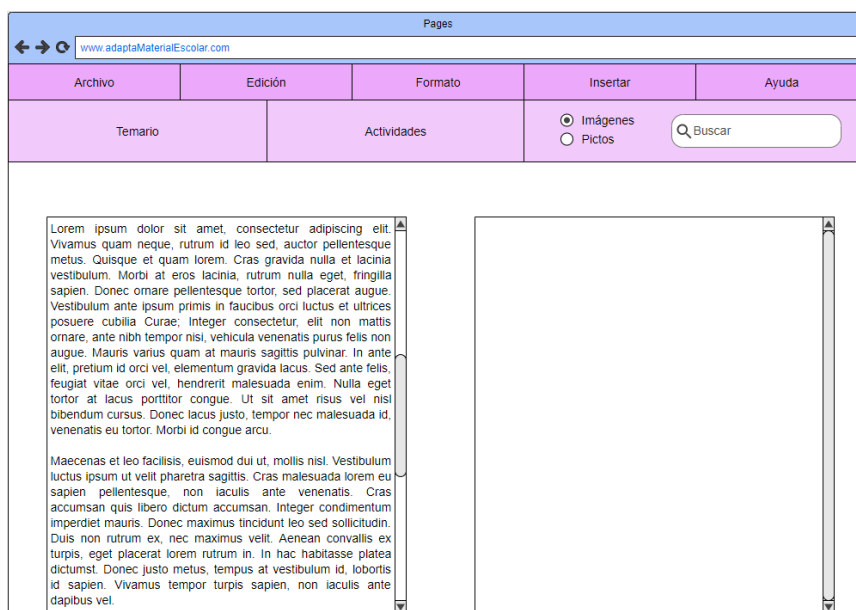


Figura 6.3: Versión 1.0 del editor

aparece una ventana con los diferentes pictogramas que se obtienen al buscar la palabra “Perro”. Previamente, para poder realizar la búsqueda de pictogramas, se ha tenido que seleccionar el campo Pictos, que está a la izquierda de la barra de búsqueda. Si se hace clic en uno de ellos automáticamente se pondrá en el campo del editor.

- **Creación de actividades:** En la Figura 6.5 se puede ver un ejemplo de creación de un ejercicio. En este caso, el ejercicio es “Completar”, es decir, completar el texto rellenando los huecos en blanco. Para ello, se ha tenido que seleccionar el menú Actividades y, de entre los distintos tipos de actividades que hay, seleccionar “Completar”. Una vez elegido este apartado, habría que ir señalando en la parte de la izquierda las palabras en las que se quiera dejar el hueco, por lo que en la parte de la derecha (editor) saldría huecos en blanco en vez de la palabra señalada. Si se quiere cerrar este menú, bastaría con darle clic en la flecha que hay en la parte de la izquierda del menú.
- **Adaptación de temario:** En la Figura 6.6 se puede encontrar un ejemplo de una adaptación de temario. En este caso, la adaptación es “Resumir texto”. Para ello, se ha tenido que seleccionar el menú Temario. Una vez elegido este apartado, lo primero es seleccionar el texto que se quiera resumir en la parte izquierda y después habría que darle clic a “Resumir texto” para que salga, en la parte de la derecha, el texto resumido.

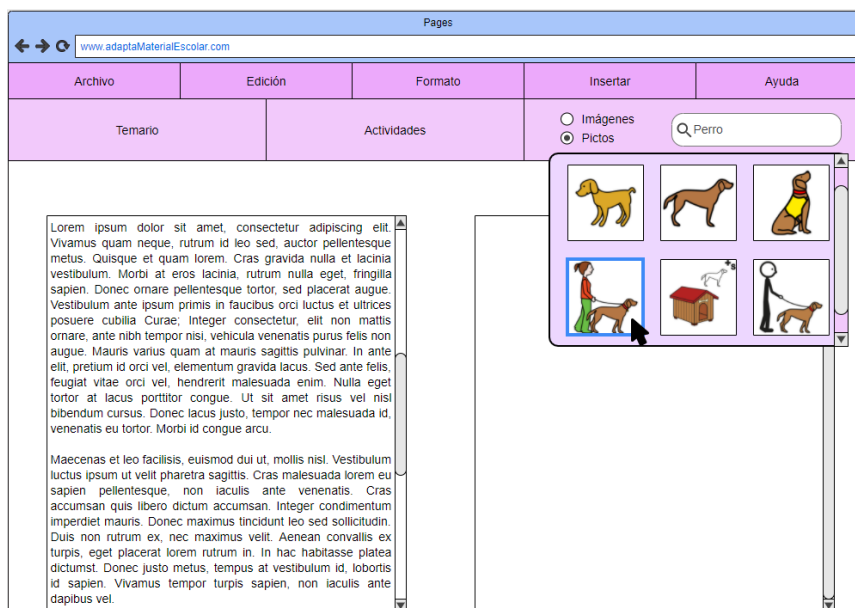


Figura 6.4: Versión 1.0 de la búsqueda de pictogramas

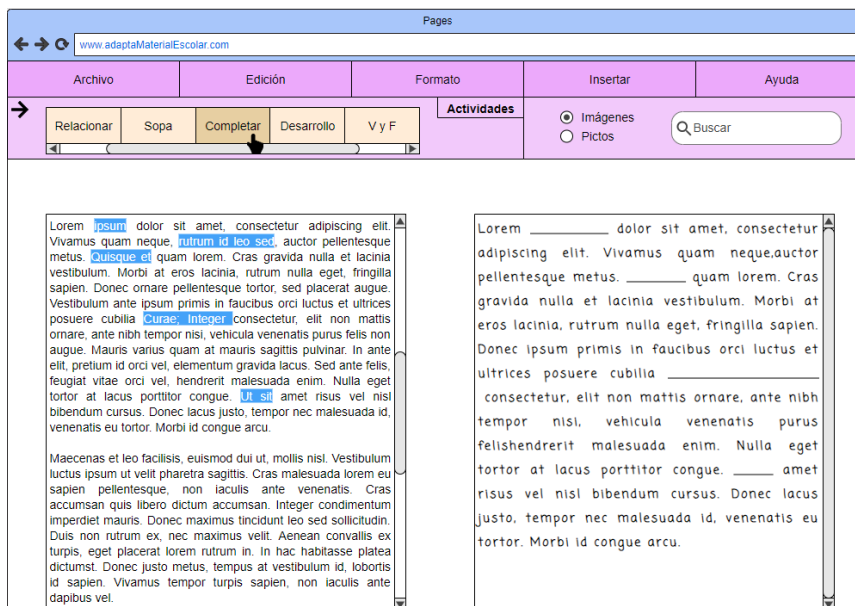


Figura 6.5: Versión 1.0 de la adaptación de actividades

6.2.3. Tercera iteración: iteración competitiva

Una vez que se hizo la versión 1.0 del prototipo, se propuso hacer una iteración competitiva para ver si este prototipo se podía mejorar. Cada miem-

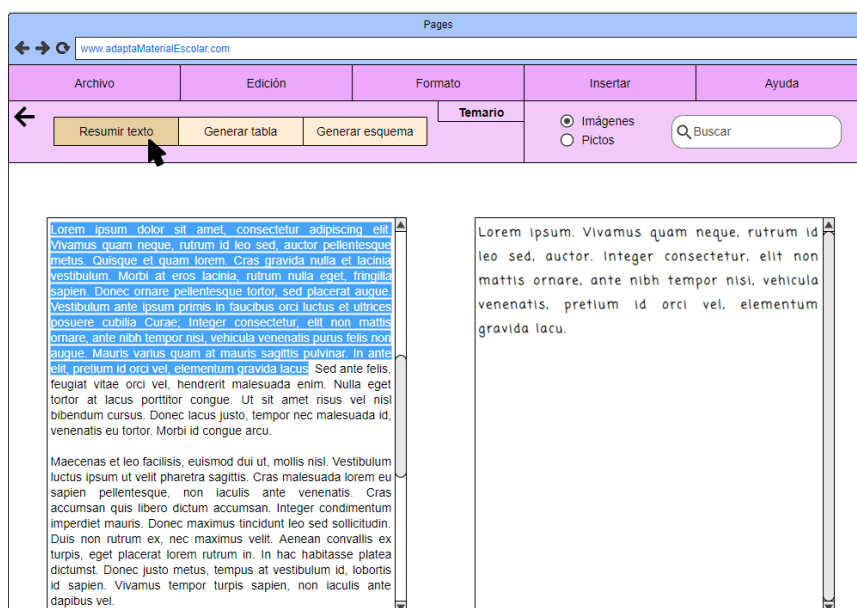


Figura 6.6: Versión 1.0 de la adaptación de temario

bro del equipo realizó, de forma individual y sin influir los unos en los otros, un prototipo propio partiendo del inicial. Con esto se consigue tener diferentes puntos de vista sobre una misma funcionalidad. Una vez que todos los prototipos fueron diseñados (dos de forma digital y otro en papel) se realizó una puesta en común para analizar las distintas propuestas. Los resultados fueron los siguientes:

- **Página principal:** En las Figuras 6.7, 6.8 y 6.9 se puede observar los resultados de los tres prototipos para la página principal. En el caso del prototipo de Jorge (Figura 6.7), en la parte inferior están los créditos y los logotipos de ARASAAC³, la Universidad Complutense de Madrid⁴ y la licencia de *Creative Commons*⁵. También se observa que en la parte de la derecha de la barra de navegación se propuso hacer un sistema de *login* para poder guardar el trabajo realizado. Ésto último no resultó debido a que los usuarios finales que usen la aplicación especificaron que no querían recordar ningún nombre de usuario y contraseña.

También, en la parte superior de dicha figura coincidimos que sería buena idea establecer algún botón, tal y como ocurre en las Figuras 6.8 y 6.9. En el primer caso, el prototipo de Natalia, se observa que hay dos botones: Créditos, donde estarían los logotipos de ARASAAC³, la Uni-

³<https://arasaac.org/>

⁴<https://www.ucm.es>

⁵<https://creativecommons.org/>

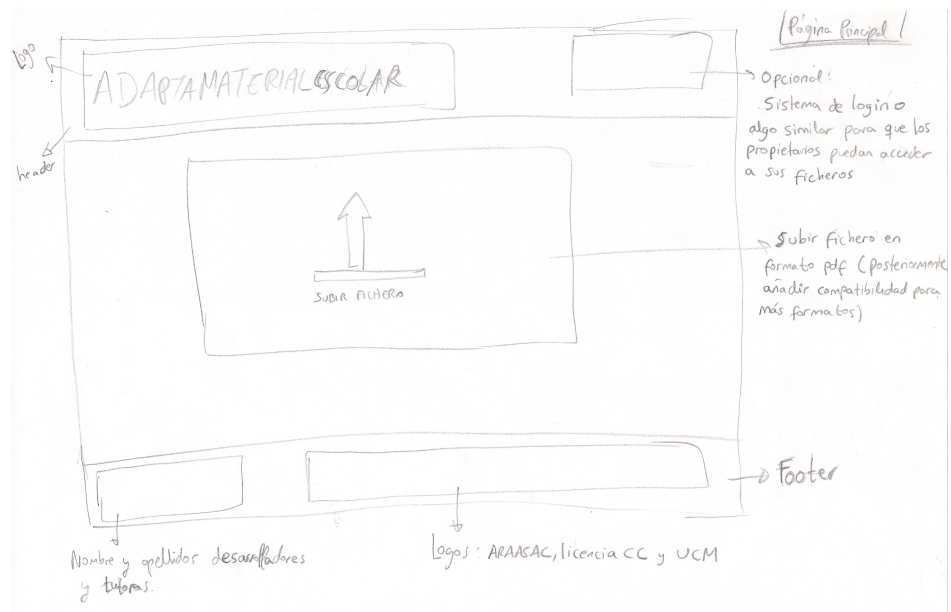


Figura 6.7: Prototipo de la página principal (Jorge)

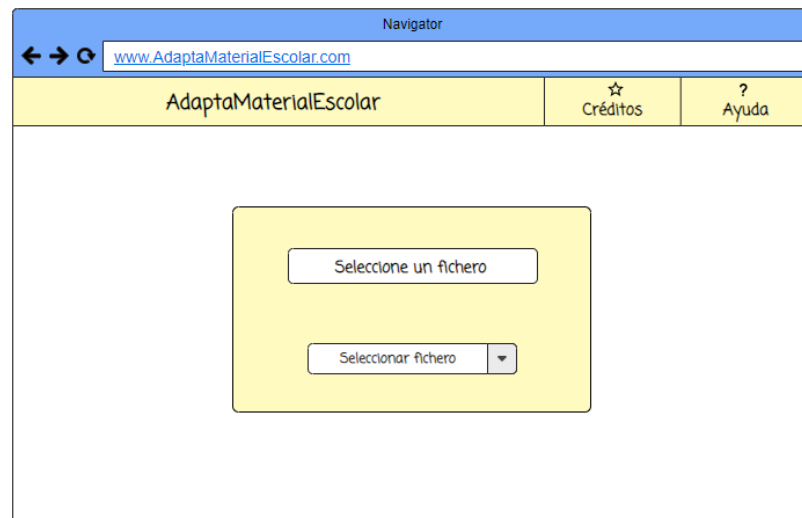


Figura 6.8: Prototipo de la página principal (Natalia)

versidad Complutense de Madrid⁴ y la licencia de *Creative Commons*⁵ y las personas que han creado el proyecto, junto con las directoras; y Ayuda, por si el usuario necesita algún tipo de ayuda con la aplicación. En el segundo caso, el prototipo de Pablo, se ven tres botones en forma de enlaces: Inicio, donde te llevaría a la página principal; Ayuda, donde se incluiría la parte de créditos y los logotipos; y Contacto, por si el



Figura 6.9: Prototipo de la página principal (Pablo)

usuario tiene algún problema y/o sugerencia pueda contactar con los desarrolladores y proporcionarle una respuesta.

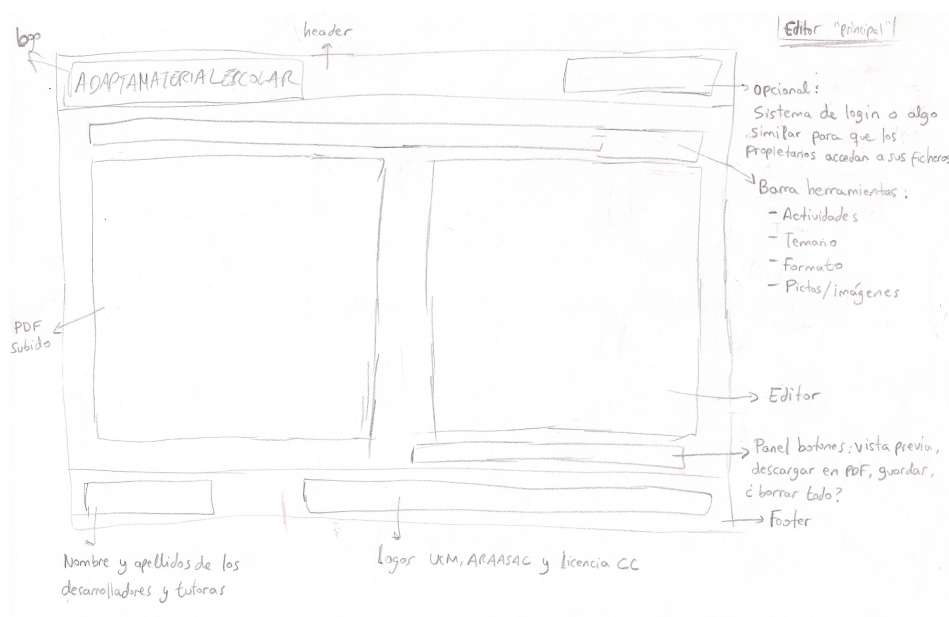


Figura 6.10: Prototipo del editor (Jorge)

- **Editor:** En las Figuras 6.10, 6.13 y 6.14 se pueden ver los resultados de la página del editor. En el caso del prototipo de Jorge (Figura 6.10), observamos una barra de navegación con diferentes menús, que representa los tipos de adaptaciones que tendrá en la aplicación (véase sección 6.1.1), junto con la búsqueda de imágenes y pictogramas. Si se

pasa el ratón por encima de uno de estos menús saldrá un desplegable con los distintos tipos de actividades, temario y formato que hay, tal y como se puede ver en la Figura 6.11.

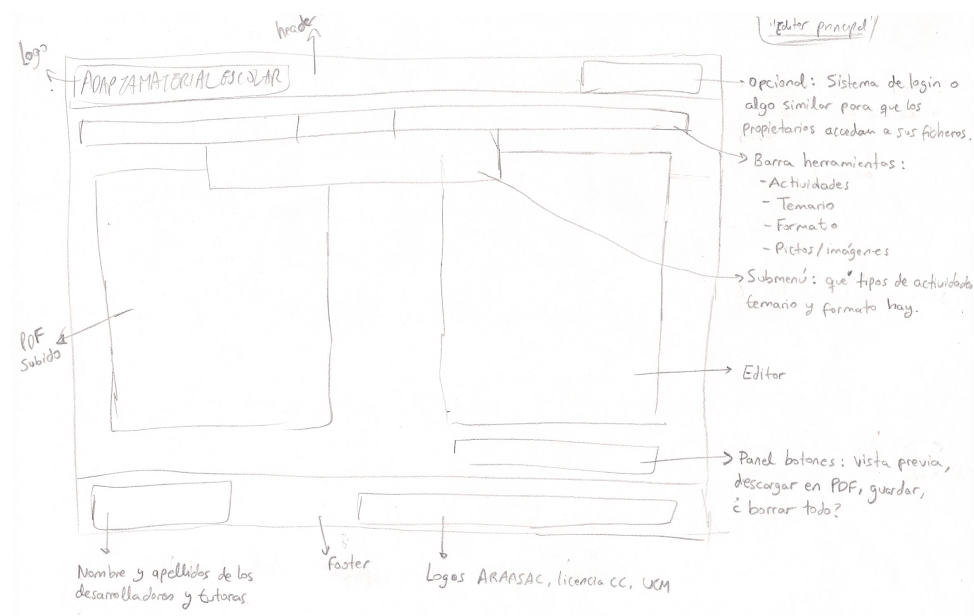


Figura 6.11: Prototipo del editor con un desplegable en uno de los menús (Jorge)

Así mismo, se ven dos elementos separados por un hueco: el elemento de la izquierda representa el fichero subido, y el de la derecha, el editor. Debajo de éste se encuentra un panel con unos botones para poder descargar, guardar y ver el documento editado, así como la posibilidad de poder borrar todo lo escrito en el editor.

En el prototipo de Natalia (Figura 6.13), también existe una barra de navegación con los mismos menús, aunque se cambia el logo de AdaptaMaterialEscolar por dichos menús, añadiendo el de Archivo y Formato. También tiene los dos elementos mencionados anteriormente: en la parte de la izquierda el documento subido, y en el de la derecha, el editor. En éste, se encuentra una barra de herramientas con la posibilidad de cambiar el formato de la letra, la posición del texto, y dos botones para descargar y borrar todo, respectivamente.

En cambio, en el prototipo de Pablo (Figura 6.14), se observa que la barra de navegación descrita en los prototipos de Jorge y Natalia no está, sino que se encuentra incluida en el propio editor (el elemento que se encuentra en la parte de la izquierda). En esta barra de herramientas se podrá dar formato al texto (ponerle forma de título, párrafo, color, negrita, cursiva, etc), así como deshacer y rehacer lo escrito, crear

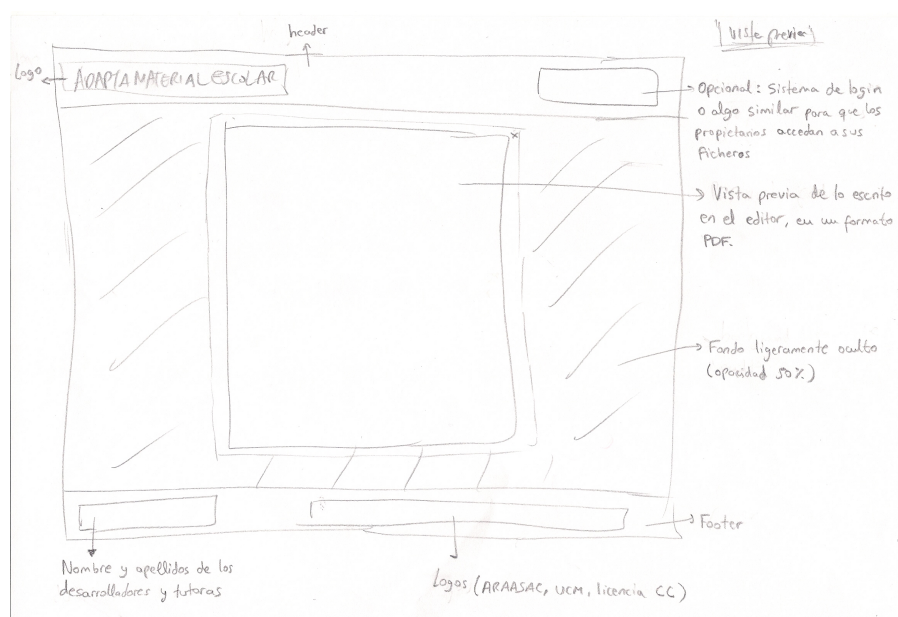


Figura 6.12: Prototipo de la vista previa (Jorge)

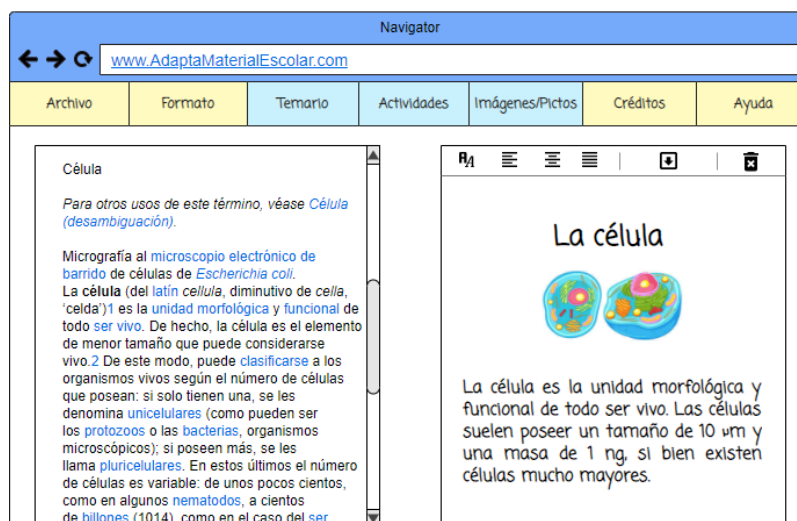


Figura 6.13: Prototipo del editor (Natalia)

tablas, y unos botones con los distintos tipos de adaptaciones que hay.

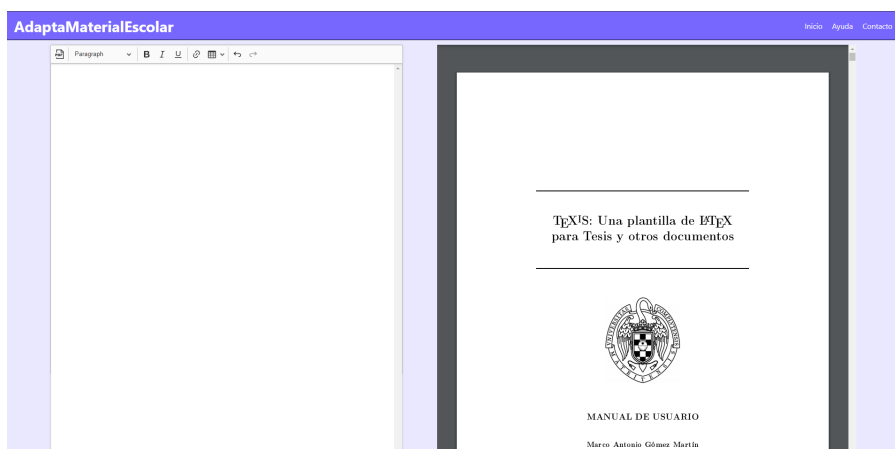


Figura 6.14: Prototipo del editor (Pablo)

6.2.4. Cuarta iteración: diseño final

Una vez hecha la puesta en común de los diseños realizados por cada miembro del equipo se procedió a hacer el diseño final. Este diseño tomaba como base el prototipo de Pablo (Figura 6.14), debido a que estaba todo más limpio y claro visualmente. A este prototipo se añadió una barra de navegación en la parte superior (como las que aparecen en los prototipos de las Figuras 6.10 y 6.13) para los tipos de adaptaciones que habrá en la aplicación. Esta barra de navegación surge porque algunas adaptaciones tienen un estilo de implementación distinto (algunas se podrán hacer sobre el propio editor pero otras abrirán nuevas ventanas para poder realizar la adaptación, sobre todo en las adaptaciones de actividades).

Tras realizar este diseño, se envió un email a las profesoras del Aula TEA para que nos dieran *feedback* sobre él. La respuesta fue positiva: tanto la distribución de los elementos como los colores usados eran adecuados.

6.3. Implementación

AdaptaMaterialEscolar es una aplicación web creada con React (explicado en la sección 4.2), cuyo código se puede encontrar en el repositorio del grupo NIL en GitHub⁶. Es accesible a través de la siguiente url: <https://holstein.fdi.ucm.es/tfg/2021/adapta/>.

⁶<https://github.com/NILGroup/TFG-AdaptaMaterialEscolar/tree/master/Codigo>

En la Figura 6.15 se muestra la página principal de la aplicación. En ella se puede ver el header en la parte superior, con el logo de la aplicación en la parte de la izquierda, y los botones de Inicio, Ayuda y Contacto en la parte de la derecha; y en el centro de la página una zona para poder subir un fichero *.pdf* que servirá como base a las adaptaciones y/o a la creación de ejercicios.

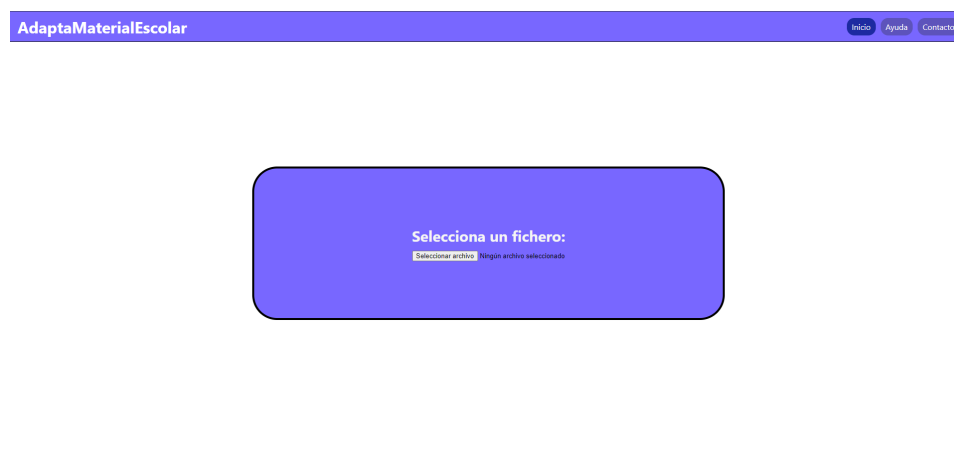


Figura 6.15: Página principal de AdaptaMaterialEscolar

Cuando el fichero se haya subido, aparecerán dos zonas diferenciadas (ver Figura 6.16): la zona de la izquierda contendrá el documento subido y la zona de la derecha es la zona de edición, donde se irá creando el nuevo documento con las adaptaciones y/o ejercicios.

Las funcionalidades implementadas se darán con más detalle en la sección 6.3.2 y son las siguientes:

- Subir un documento fuente. Permite subir un documento en formato *.pdf* que podrá servir de texto base para realizar las adaptaciones y/o para la creación de ejercicios. En la Figura 6.15 se puede ver la página de subir un documento fuente.
- Editor. Permite escribir, modificar las adaptaciones y/o ejercicios ya escritos, cambiar el formato (tipo, tamaño, color de letra, posicionamiento del texto...), etc. En la Figura 6.16 se puede ver el editor y la barra de herramientas que contiene las opciones de formato.
- Pictogramas. Permite buscar pictogramas a partir de una palabra. La búsqueda devolverá varios pictogramas y haciendo clic en un pictogra-

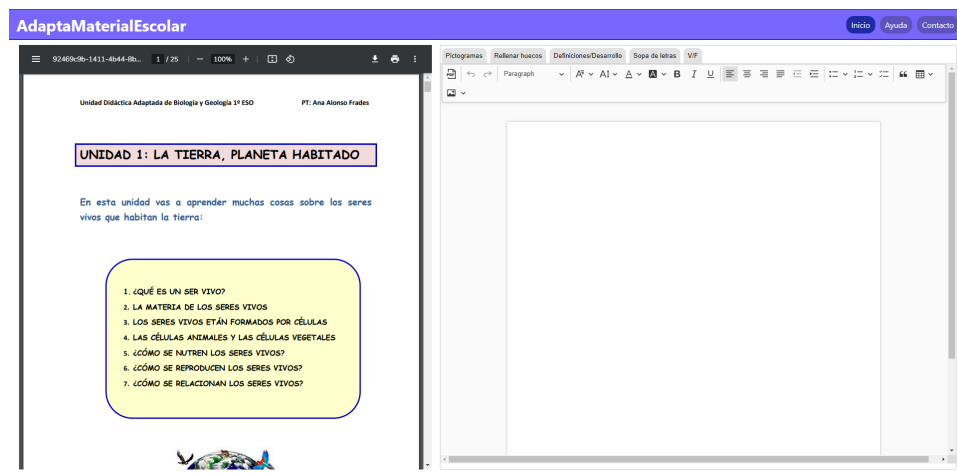


Figura 6.16: Página principal de AdaptaMaterialEscolar tras subir un documento

ma éste será llevado al editor. En la Figura 6.17 se puede ver la interfaz del buscador de pictogramas cuando se ha buscado la palabra flor.

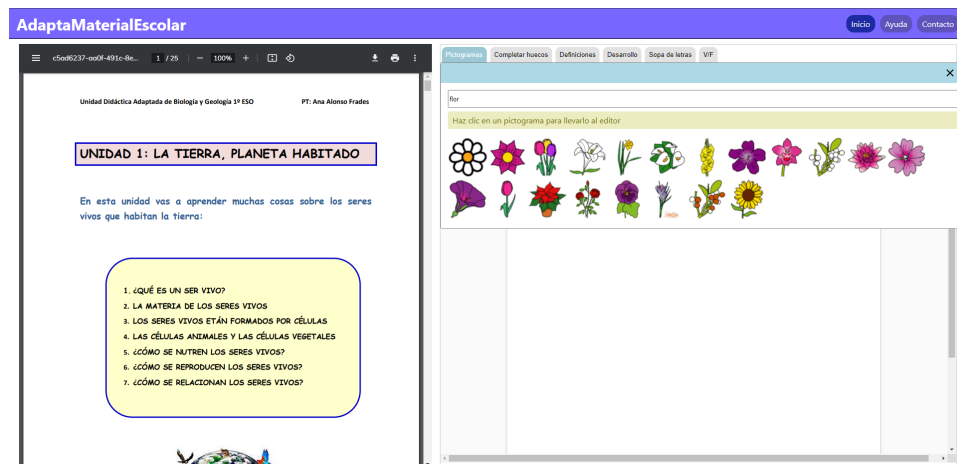


Figura 6.17: Interfaz de AdaptaMaterialEscolar al seleccionar Pictogramas

- **Completar huecos.** Permite crear un ejercicio de completar huecos. Primero habrá que introducir el texto base del ejercicio y posteriormente seleccionar las palabras en las que se quiera dejar el hueco. En la Figura 6.18 se muestra la interfaz la aplicación al seleccionar Completar huecos.
- **Definiciones.** Permite crear una lista de conceptos a definir en un número máximo de líneas. Incluye una opción para añadir espaciado extra

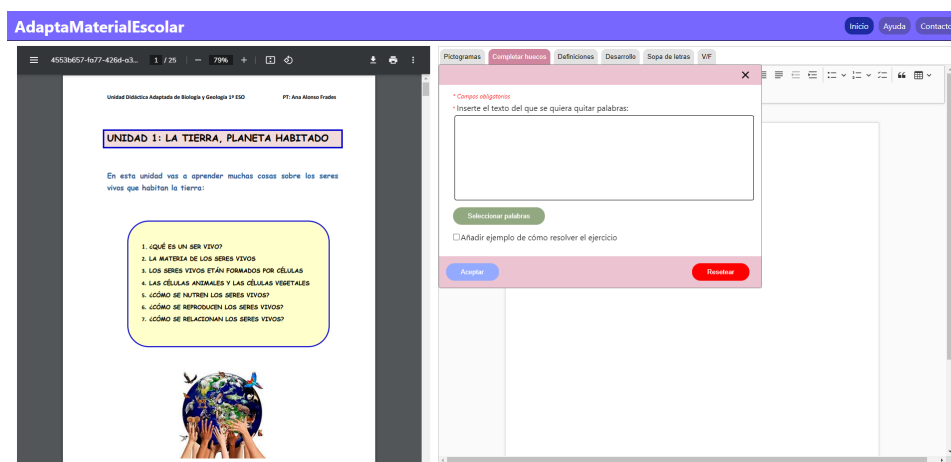


Figura 6.18: Interfaz de AdaptaMaterialEscolar al seleccionar Completar huecos

entre líneas para estudiantes que tengan una letra más grande. En la Figura 6.19 se puede ver la interfaz de la aplicación cuando se ha seleccionado Definiciones.

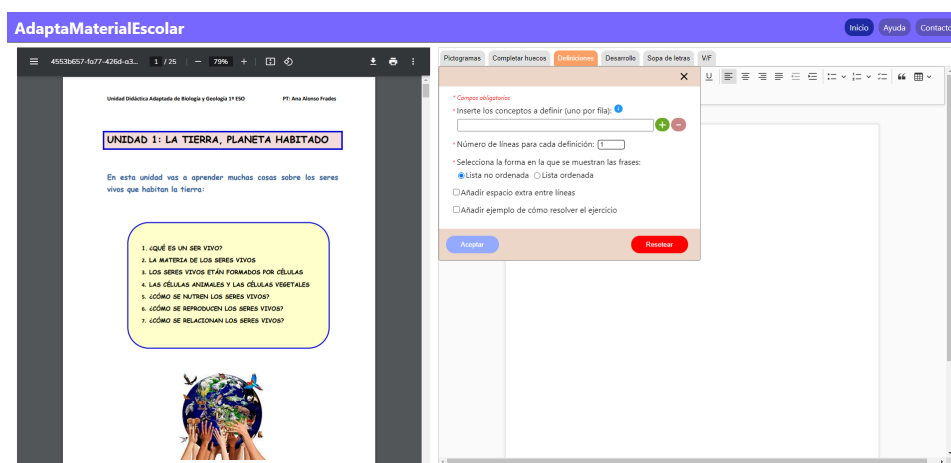


Figura 6.19: Interfaz de AdaptaMaterialEscolar al seleccionar Definiciones

- Desarrollo. Permite crear un ejercicio de desarrollo insertando el enunciado y el número de líneas máximo para responder. También tiene una opción para añadir espaciado extra entre líneas para estudiantes que tengan una letra más grande. En la Figura 6.20 se puede ver la interfaz de la aplicación cuando se ha seleccionado Desarrollo.
- Sopa de letras. Permite crear una sopa de letras estableciendo el número

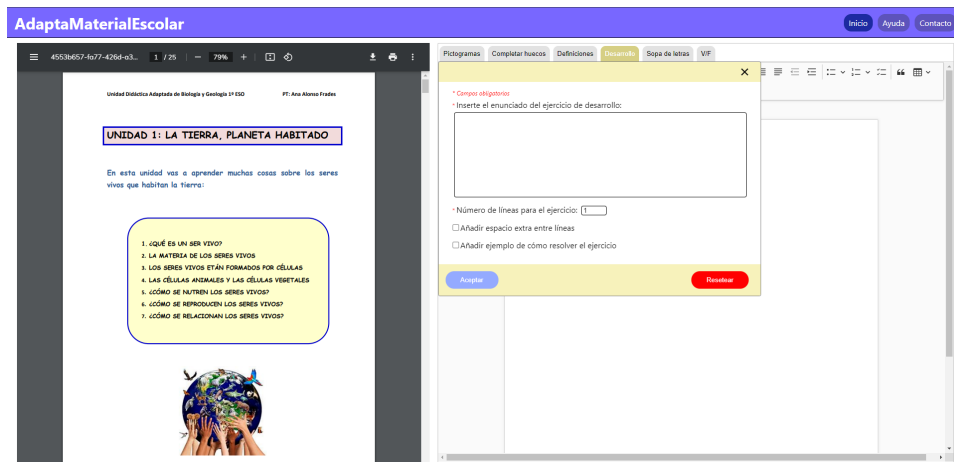


Figura 6.20: Interfaz de AdaptaMaterialEscolar al seleccionar Desarrollo

de filas y columnas, las palabras que se quieren insertar en la sopa de letras y las direcciones cardinales para buscar la palabra. Incluye una opción para activar la escritura al revés, donde cada palabra tiene una probabilidad de escribirse al revés, establecida por el usuario, y también una opción para mostrar junto al enunciado las palabras a buscar si el usuario ha activado esta opción. En la Figura 6.21 se puede ver la interfaz de la aplicación al seleccionar Sopa de letras.

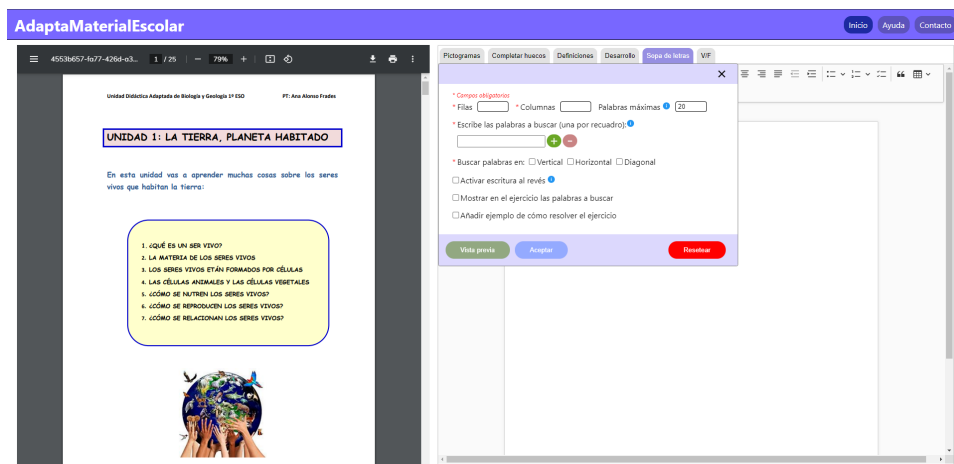


Figura 6.21: Interfaz de AdaptaMaterialEscolar al seleccionar Sopa de letras

- Verdadero o falso. Permite crear una lista de oraciones y añade al final un recuadro para contestar con una V si es verdadero o con una F si es falso. En la Figura 6.22 se puede ver la interfaz de la aplicación al seleccionar V/F.

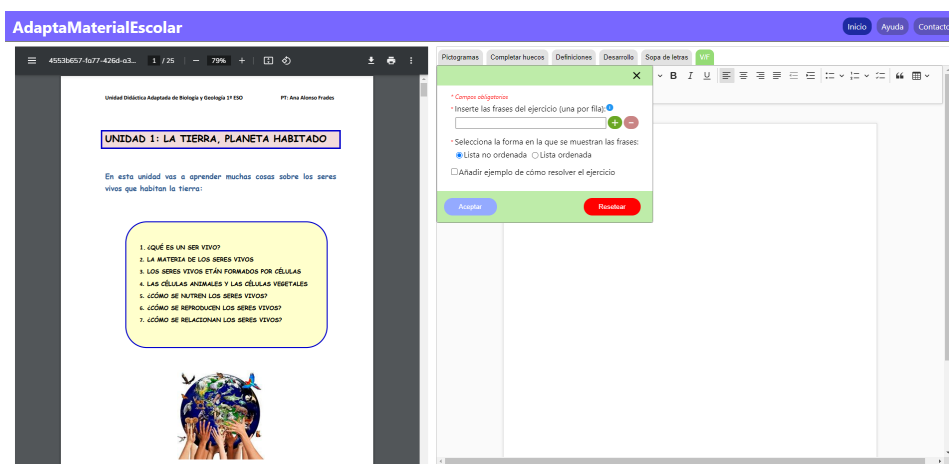


Figura 6.22: Interfaz de AdaptaMaterialEscolar al seleccionar V/F

Todos los ejercicios tienen una opción para añadir un ejemplo de cómo resolver el ejercicio, y en todos ellos se añade un enunciado plantilla automáticamente, que posteriormente el usuario puede personalizar a su gusto. En las Figuras 6.36, 6.40, 6.44, 6.48 y 6.54 se observa el resultado de todos los ejercicios, donde se añade el enunciado y un párrafo sobre cómo resolver el ejercicio.

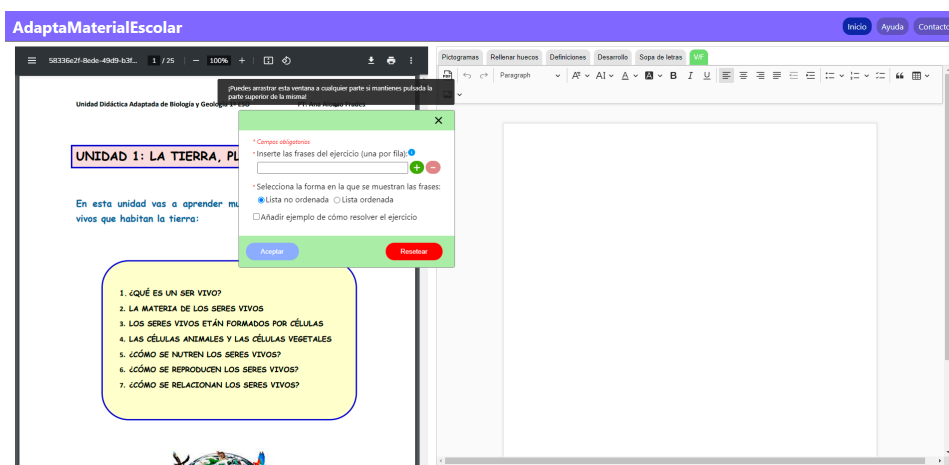


Figura 6.23: Ejemplo de ventana movable

Al hacer clic en una de las funcionalidades, se abrirá, justo debajo de la barra de funcionalidades, una ventana movable manteniendo el clic izquierdo pulsado en la parte superior de dicha ventana. En la Figura 6.23 se puede ver el mensaje que sale al pasar el ratón por encima de la parte superior de la ventana en el que informa al usuario de que puede moverla; y se ve que

ésta se ha movido de sitio.

6.3.1. Arquitectura

En el proyecto de este TFG se ha decidido aplicar una arquitectura *Redux*⁷. Redux es un patrón de arquitectura de datos que pretende disminuir el número de relaciones entre los componentes de la aplicación. Se trata de un patrón que se adapta a todo tipo de librería o *framework* del lado del cliente. La Figura 6.24 se puede ver un diagrama mostrando el flujo de una aplicación Redux. Dicho diagrama muestra un flujo unidireccional, donde las vistas se suscriben a los datos del Store y actualizan estos mediante acciones.

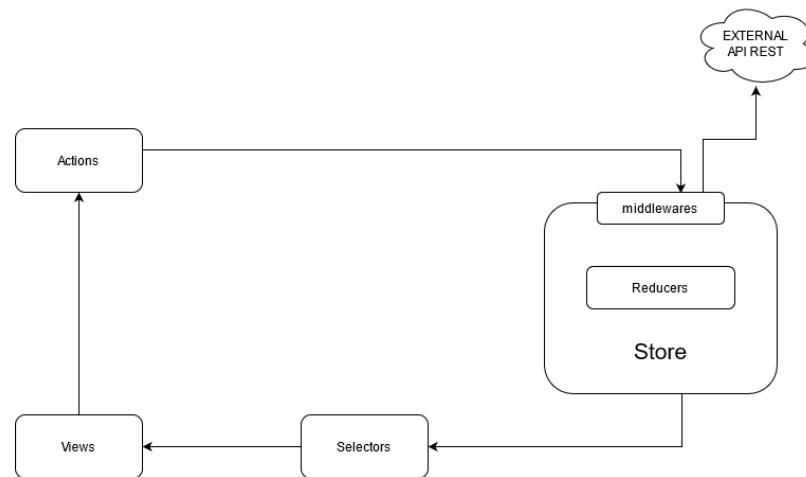


Figura 6.24: Diagrama de flujo en Redux

Esta arquitectura se basa en tres principios:

- Única fuente de la verdad. El estado de la aplicación se guarda en un único Store y cuando los componentes necesitan conocer el estado lo cogerán del Store. Este Store es un objeto Javascript y está estructurado en forma de árbol con todos los datos necesarios para la aplicación.
- El estado es de solo lectura. No podemos modificar el estado directamente, solo podremos leer de él para mostrarlo en la vista y si queremos modificarlo tendremos que ejecutar acciones. Las acciones son un objeto Javascript que incluye al menos un atributo *type* que indica el tipo

⁷<https://redux.js.org/>

de acción que estamos emitiendo y, en caso de que haya datos asociados al cambio o modificación, un atributo *payload* con esos datos. Por ejemplo, en el siguiente fragmento de código se tiene una acción para cargar un documento, donde el atributo *type* indica el tipo de acción a realizar y el *payload* llevará los datos del documento a cargar:

```
export const loadDocument = (document) => ({
  type: DocumentActionTypes.LOAD_FILE,
  payload: document
});
```

Listing 6.1: Ejemplo de acción para cargar un documento

- Cambios con funciones puras. Ya que el estado no se puede modificar directamente y está almacenado en un Store, tenemos que ejecutar acciones puras llamadas *reducers* para modificarlo. Un *reducer* es una función que recibe dos parámetros, el estado inicial y una acción. Dentro contendrá un *switch-case* que, en base a la acción que reciba, ejecutará los cambios necesarios en el Store. Por ejemplo, en caso de recibir una acción del tipo `LOAD_FILE`, almacenará el documento recibido del *payload* de la acción y lo guardará en el Store. El siguiente fragmento muestra un reducer del módulo documentos donde gestiona la acción `LOAD_FILE`:

```
const documentReducer = (state = INITIAL_STATE,
  action) => {
  switch (action.type){
    case DocumentActionTypes.LOAD_FILE:
      return {
        ...state,
        file: action.payload.file,
        fileIsLoaded: true
      };
    default:
      return state;
  }
};
```

Listing 6.2: Ejemplo de reducer donde se gestiona el tipo de acción `LOAD_FILE`

Al externalizar toda la información en un Store, facilitamos el acceso y modificación de datos en la aplicación. En la Figura 6.25 se muestran las diferencias de flujo de información entre los componentes de una aplicación web sin Redux y otra con Redux.

En Redux, el flujo de datos es unidireccional (ver Figura 6.24) y se siguen los siguientes pasos:

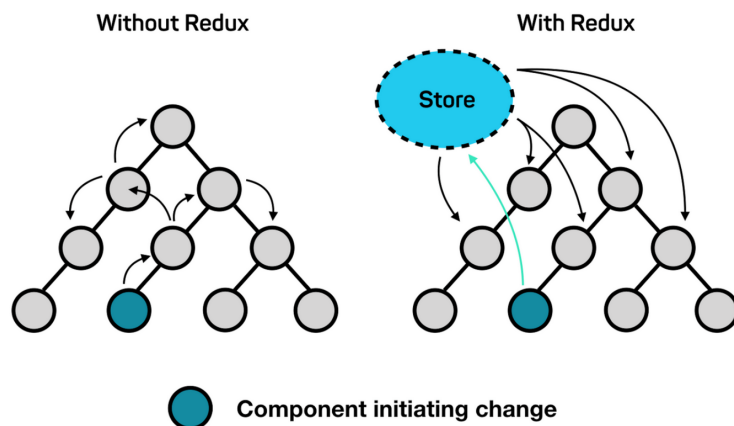


Figura 6.25: Comparación de flujo de información de componentes sin y con Redux. Fuente: <https://blog.codecentric.de/en/2017/12/developing-modern-offline-apps-reactjs-redux-electron-part-3-reactjs-redux-basics/>

- Los componentes se suscriben a los datos del Store mediante selectores. Estos selectores actúan como *getters*, es decir, funciones de solo lectura de datos. Cuando el Store modifique uno de esos valores, los componentes a los que estén suscritos se actualizarán automáticamente. Se recurre a una librería de npm denominada *reselect*⁸ que simplifica estas implementaciones. Mediante *reselect* ejecutamos una función denominada *createSelector*, que internamente realiza la suscripción al dato del Store que queramos. En el siguiente fragmento se crea un selector para obtener el documento cargado en el Store. Se selecciona el módulo *document* del Store y creamos un selector que obtenga el fichero:

```
const selectDocumentStore = (store) => store.
  document;

export const selectDocumentFile = createSelector(
  [selectDocumentStore],
  documentStore => documentStore.file
);
```

Listing 6.3: Ejemplo de selector para obtener el documento cargado en el Store

En los componentes se inyectarán los selectores necesarios mediante una función llamada *createStructuredSelector*. Por ejemplo, en el componente que muestra un documento:

```
const mapStateToProps = createStructuredSelector({
```

⁸<https://www.npmjs.com/package/reselect>

```
document: selectDocumentFile
});
```

Listing 6.4: Ejemplo de inyección del selector en el componente

- Los componentes ejecutan acciones. Esto lo realizan ejecutando un método *dispatch* y añadiendo como parámetro la acción a ejecutar. Por ejemplo, si tenemos una acción para cargar un documento, haríamos lo siguiente:

```
export const loadDocument = document => ({
  type: DocumentActionTypes.LOAD_FILE,
  payload: document
});
```

Listing 6.5: Ejemplo de acción para cargar un documento

- Se ejecutan *middlewares* en caso de necesitar acciones asíncronas. Estos *middlewares* permiten ejecutar llamadas a API REST y posteriormente despachar otras acciones según el resultado. El siguiente ejemplo muestra cómo se ejecuta un *middleware* de buscar pictogramas llamando a la API de ARASAAC:

```
export const fetchPictogramsAction = (searchInput)
=> {
  return dispatch => {
    fetch("https://api.arasaac.org/api/pictograms/es
/search/${searchInput}")
    .then((response) => response.json())
    .then(data => {
      let items = [];
      for(let i = 0; i < data.length && i < 20; i++)
      {
        items.push("https://static.arasaac.org/
pictograms/${data[i].\_id}/${data[i].\_id
}\_500.png")
      }
      dispatch(fetchPictogramsSuccess(items));
    })
    .catch(error){
      dispatch(fetchPictogramsFailure(error))
    };
  }
}
```

Listing 6.6: Ejemplo de ejecución de un middleware para buscar pictogramas llamando a la API de ARASAAC

- El Store de Redux invoca a la función reductora solicitada por la acción. El Store le pasará dos argumentos a la función: estado actual y acción. El *switch* dentro del reductor encontrará la acción solicitada y

actualizará los datos en base a ella. Los selectores detectarán el cambio y actualizarán las vistas.

Redux lo hemos usado en el proyecto para reducir la interacción entre los distintos componentes de la aplicación, simplificando así la complejidad y facilitando el mantenimiento y depuración.

6.3.2. Funcionalidades

En esta sección se explicarán con detalle la implementación de cada una de las funcionalidades.

6.3.2.1. Subir un documento fuente

En la Figura 6.15 se puede ver el aspecto de AdaptaMaterialEscolar cuando se tiene que subir un documento fuente. El documento a subir debe ser en formato *.pdf*. En el caso de que no se suba en dicho formato, el fichero no se renderizará correctamente.

Una vez que se ha seleccionado el fichero *.pdf*, para que se cargue en la aplicación, ocurre lo siguiente (en la Figura 6.26 se muestra el diagrama de secuencia de esta funcionalidad):

1. La función *uploadFile()* hace una llamada al despachador pasándole como parámetro *loadDocument(document)*, donde *document* es el objeto que contiene el documento a subir.
2. El despachador buscará en *Actions* el tipo de acción que es (en este caso, *LOAD_FILE*), junto con su *payload* (el documento a subir) y lo devolverá. Posteriormente, llamará al reductor a través de *documentReducer()* pasándole como parámetros el estado (*state*) del componente y la acción devuelta anteriormente. A continuación, buscará en el *switch* el tipo de acción y procederá a modificar el estado del componente, actualizando los campos *fileL* (el documento subido) y *fileIsLoaded* (indica si hay un documento cargado, y sirve para cambiar entre las páginas de subir un documento si su valor es *false*, y del editor, si su valor es *true*). Tras actualizarlos, devolverá el estado modificado.
3. Por último, tras realizar este proceso, el documento se habrá cargado en la aplicación, como se puede ver en la Figura 6.16. El texto del

archivo subido se podrá usar de base y se podrá utilizar el editor para realizar adaptaciones de temario y crear ejercicios.

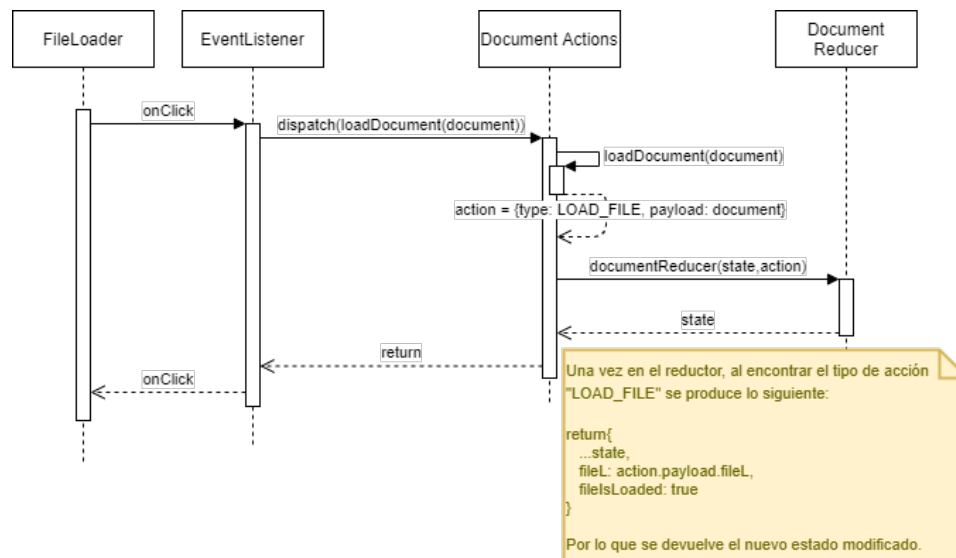


Figura 6.26: Diagrama de secuencia de subir un documento

En la Figura 6.27 se puede ver un diagrama con el flujo de acción, reductor, etc. concretos para esta función.

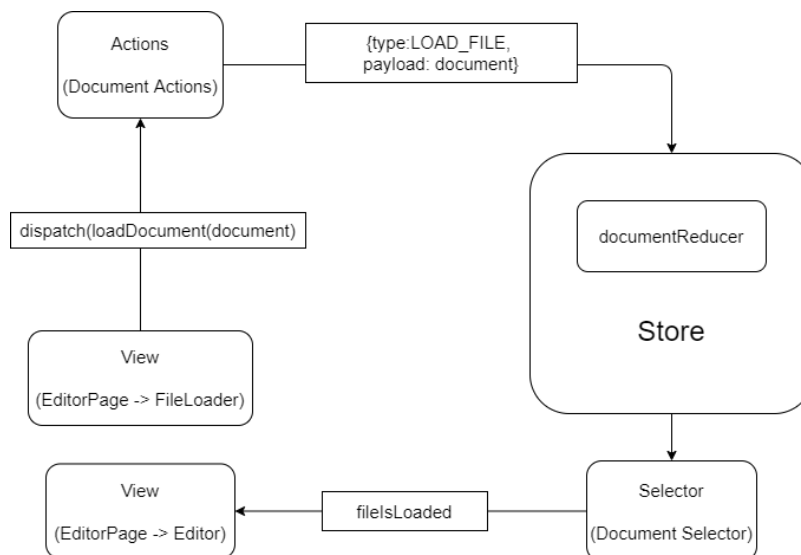


Figura 6.27: Diagrama de flujo de subir un documento

6.3.2.2. Editor

El editor de texto que hemos usado es *CKEditor 5* (ver sección 4.5). En la Figura 6.16 se puede ver, en la parte de la derecha, el editor. Para añadir funcionalidad al éste, se necesitan incluir *plugins*⁹. Estos *plugins* pueden ser del propio editor o creados por un usuario y la mayoría tienen configuración personalizable. El editor consta de dos partes: la barra de herramientas y la zona de escritura. La barra de herramientas consta de los siguientes elementos:

- Exportar a PDF: permite exportar a un documento *.pdf* el contenido del editor. Es posible gracias al *plugin ExportPdf*.
- Deshacer y rehacer: permite deshacer/rehacer la última acción realizada en el editor. Es posible gracias al *plugin Essentials*.
- Estilos de párrafo: permite cambiar el estilo del párrafo. Es posible gracias al *plugin Heading*. Se pueden definir nuevos estilos y añadirlos a la configuración del *plugin*.
- Fuente de la letra: permite cambiar la fuente de la letra. Es posible gracias al *plugin FontFamily*. Cada familia de fuente de texto que se quiera incluir debe ser añadida a las opciones de configuración del *plugin*.
- Tamaño de la letra: permite cambiar el tamaño de la letra. Es posible gracias al *plugin FontSize*. Se pueden definir tamaños de letra personalizados añadiéndolos a las opciones de configuración del *plugin*.
- Color de la letra: permite cambiar el color de la letra. Es posible gracias al *plugin FontColor*. Se pueden definir colores personalizados y añadirlos a las opciones de configuración del *plugin*, así como el número de columnas a mostrar.
- Color de resaltado: permite resaltar una palabra con un color. Es posible gracias al *plugin FontBackgroundColor*. Se pueden definir colores personalizados y añadirlos a las opciones de configuración del *plugin*, así como el número de columnas a mostrar.
- Negrita, cursiva y subrayado: permite cambiar el formato de texto a negrita, cursiva y/o subrayado gracias a los *plugins Bold*, *Italic* y *Underline*, respectivamente.

⁹<https://ckeditor.com/docs/ckeditor5/latest/builds/guides/development/plugins.html>

- Alineamiento del texto: permite alinear el texto a la izquierda, centrarlo, alinearlo a la derecha o justificarlo. Es posible gracias al *plugin Alignment*.
- Sangría: permite disminuir o aumentar la sangría de un párrafo. Es posible gracias al *plugin Indent* e *IndentBlock*.
- Listas: permite crear listas no numeradas, listas numeradas y listas de tareas. Es posible gracias a los *plugins ListStyle*, para las listas numeradas y no numeradas, y *ToDoList* para las listas de tareas.
- Cita en bloque: permite insertar citas en bloque. Es posible gracias al *plugin BlockQuote*.
- Tablas: permite insertar tablas y modificar el diseño de las mismas (mezclar celdas, añadir o eliminar filas y columnas, cambiar de color la tabla y celdas, etc.). Es posible gracias a los *plugins Table*, *TableToolbar*, *TableProperties* y *TableCellProperties*.
- Imágenes: permite insertar imágenes a través de una URL y redimensionarlas. Es posible gracias a los *plugins Image*, *ImageInsert*, *ImageToolbar*, *ImageStyle* e *ImageResize*.

El editor está configurado por defecto con la fuente de letra Arial, un tamaño de letra de 16px y el texto alineado a la izquierda, siguiendo las recomendaciones de Lectura Fácil ya comentadas en la sección 3.1.1.

Para que un ejercicio se pueda escribir en el editor, es necesario crear el *plugin* del ejercicio correspondiente. Éste debe tener dos ficheros: *insert-ComponenteCommand*, el fichero que contiene la ejecución del comando, y *componentePlugin*, el fichero que contiene la creación del comando y su adición a la lista de comandos del editor. En el primer fichero se debe definir la función *execute()*, cuyo parámetro son los datos que se necesitan para visualizar el ejercicio (puede ser un objeto, una sola variable, etc.). En el segundo fichero se debe definir el comando de inserción y añadirlo a la lista de comandos del editor. En el siguiente fragmento de código se muestra un ejemplo del fichero *plugin* del ejercicio de completar huecos, donde se define el comando y se añade a la lista de comandos del editor.

```
init(){  
    this.editor.commands.add('insertFillGaps', new  
        InsertFillGapsCommand(this.editor));  
}
```

Listing 6.7: Ejemplo del fichero plugin del ejercicio de completar huecos donde se define y añade el comando

En el siguiente fragmento de código se muestra un ejemplo de la función *execute()* del fichero inserción del comando del ejercicio de completar huecos.

```
execute(data){
  this.editor.model.change(writer => {
    let text = "";
    data.text.forEach(t =>{
      text += t + " ";
    })
    let insertPosition = this.editor.model.document.
      selection.getFirstPosition();
    const enunciado = writer.createElement('paragraph',
      insertPosition);

    writer.insertText(text, enunciado);
    this.editor.model.insertContent(enunciado);
    if(data.addHowToSolve){
      const howTo = writer.createElement('paragraph');
      writer.insertText("Cómo resolver el ejercicio: ... ",
        howTo, "end");
    }
    this.editor.execute("shiftEnter");
  });
}
```

Listing 6.8: Ejemplo del fichero inserción del comando del ejercicio de completar huecos donde se define la función *execute()*

La Figura 6.28 muestra un diagrama de secuencia de cómo se inicializa el editor y cómo se ejecuta la función *execute()* al ser llamado por un componente cualquiera. Para ello, hay dos fases a ejecutar. En la primera fase se inicializa el editor y nos centraremos en la inicialización de cada *plugin* y comando:

1. Cuando se crea el editor se llama a la función *create()* pasándole como parámetros *sourceElementOrData*, el componente HTML donde estará montado el editor; y *config*, el objeto de configuración del editor en el que se encuentra el idioma del editor, los plugins y algunas configuraciones de éstos. Dentro de esta función, se crea el editor haciendo un *new this()* y pasándole como parámetros los mismos que en el *create*, y asignará este editor a una variable. Posteriormente, se llama a la función *initPlugins()* del editor, pasándole como parámetro los *plugins* del objeto *config*.
2. La llamada a *initPlugins()* iterará sobre cada *plugin* que haya en *plugins*. Para ello, se hace una llamada a *init()* pasándole como parámetros los *plugins*.
3. La llamada a *init()* se ejecuta en la clase *PluginsCollection*, una clase

de *CKEditor 5* para gestionar los *plugins*. A continuación se ejecuta un bucle *forEach* en el que, para cada *plugin* de *plugins*, se hará una llamada a *init()*. Esta llamada a *init()* la tiene cada clase *ComponentePlugin* obligatoriamente.

4. La función *init()* de cada clase *ComponentePlugin* insertará, en la lista de comandos del editor, el comando correspondiente a esa clase. Se hace una llamada a *add()* pasándole como parámetros el nombre del comando y un objeto de la clase *InsertComponentCommand()* donde se encuentra la función *execute()*. La llamada a *new InsertComponentCommand()* necesita como parámetro el editor.
5. Una vez hecho todo este proceso, devolverá el editor inicializado. Hay que tener en cuenta que en todo este proceso se llaman a más funciones y se asignan más variables, ya que lo hace internamente y no son funciones que nosotros hayamos creado.

En la segunda fase nos centraremos en la ejecución de la función *execute()* de un componente cualquiera:

1. Un componente hace una llamada a la función *execute()* del editor pasándole como parámetros el nombre del comando "*insertComponente*" y el objeto con los datos necesarios para la visualización del componente en el editor, *options*.
2. La función *execute()* del editor ejecuta un bloque try-catch:
 - a) Try: se hace una llamada a *execute()* de la clase *CommandCollection*, una clase de CKEditor para gestionar los comandos, pasándole como parámetros el nombre del comando y el objeto de datos. Si todo va bien, se obtendrá el comando y se llamará a la función *execute()* de la clase *InsertComponentCommand* pasándole como parámetro el objeto de datos. En el caso de que el comando no exista se lanzará una excepción.
 - b) Catch: si ha habido alguna excepción y no se ha conseguido ejecutar el comando, se capturará la excepción y se relanzará otra excepción llamando a la función *rethrowUnexpectedError()* pasándole como argumentos el error y el editor.
3. Si todo ha ido bien, se visualizará en el editor el componente.

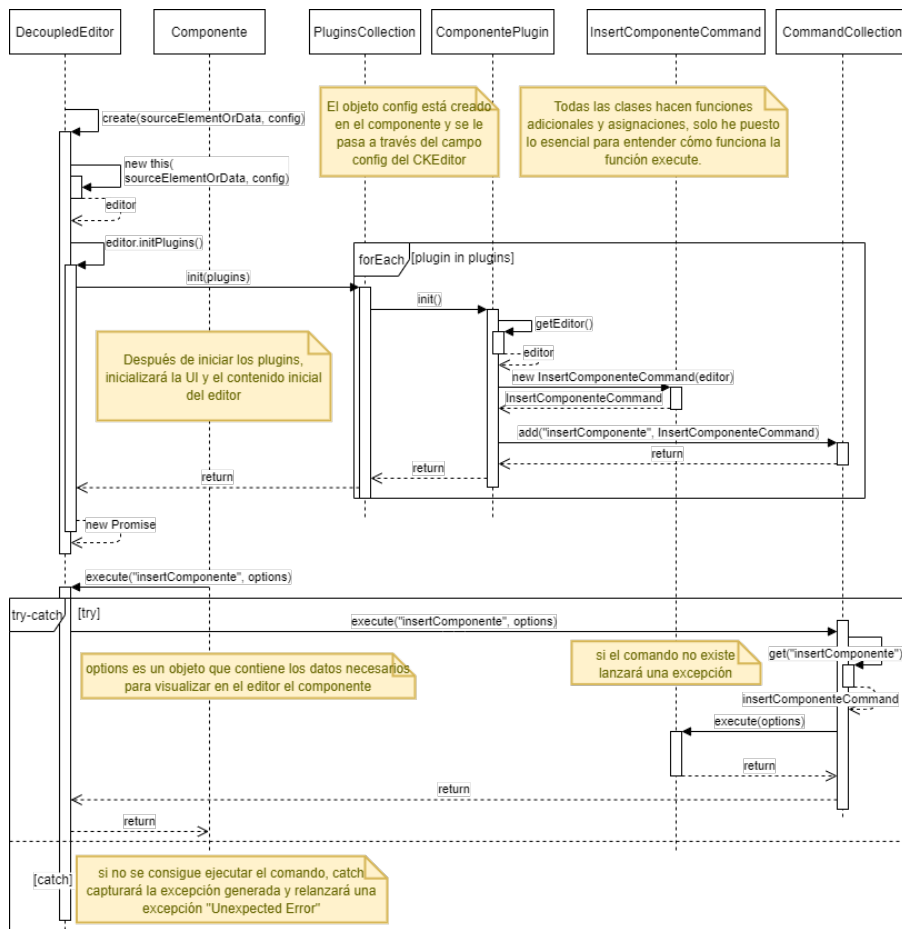


Figura 6.28: Diagrama de secuencia del proceso de inicialización del editor y la ejecución de la función execute()

6.3.2.3. Buscador de pictogramas

La Figura 6.17 muestra la apariencia de la aplicación cuando se selecciona el buscador de pictogramas. En él, se muestra un único input donde se introduce el texto a buscar. La aplicación reaccionará automáticamente y mostrará los resultados conforme se vaya actualizando el texto a buscar. En la Figura 6.29 se puede ver cómo se muestran los resultados una vez se ha introducido un texto de búsqueda.

Una vez se ha introducido el texto, se selecciona el pictograma deseado y se hace clic sobre él para insertarlo donde esté el cursor en el editor. La Figura 6.30 contiene el aspecto del editor una vez se ha introducido un pictograma.

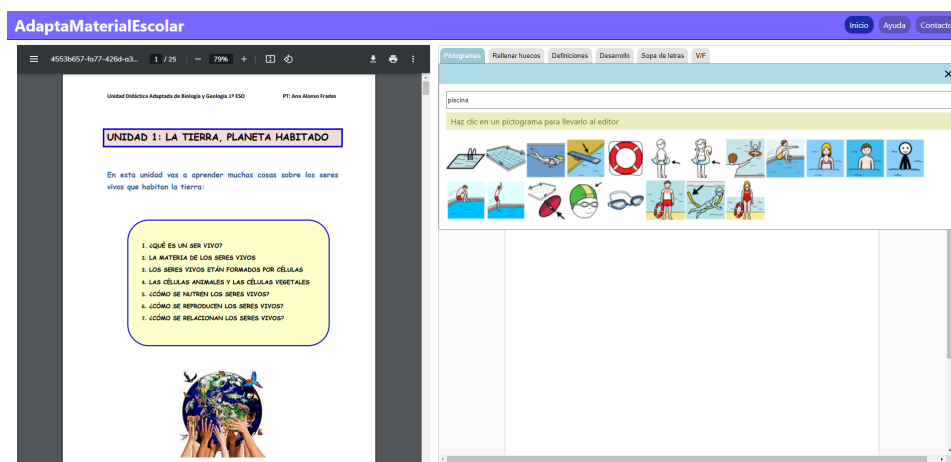


Figura 6.29: Resultados del buscador de pictogramas

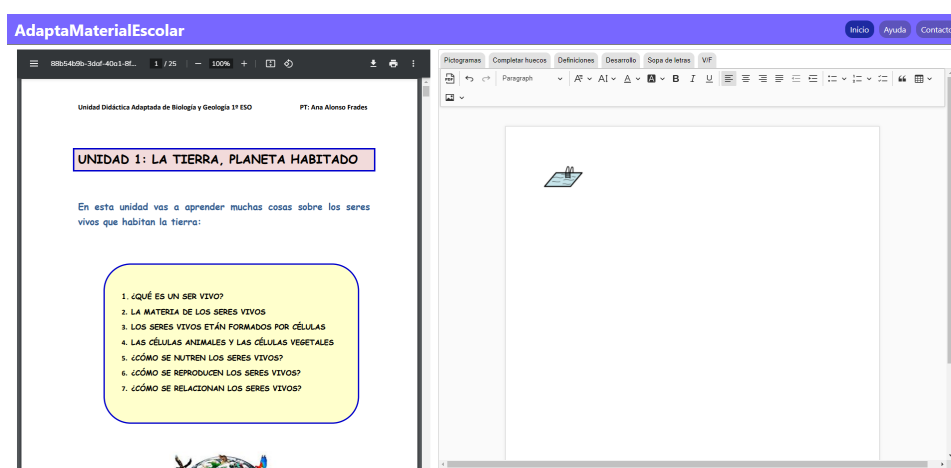


Figura 6.30: Pictograma introducido en el editor

Las Figuras 6.31 y 6.32 muestran el diagrama de clase y secuencia asociados a este caso de uso. Internamente, este módulo contiene tres valores: `searchResults`, `inputText` y `errorText`. El `inputText` contiene el texto del formulario de búsqueda. Cada vez que se actualice la vista *PictogramSearch-Modal* se ejecutará una acción *FetchPictogramsAction*, la cual realizará la llamada a la API externa de ARASAAC y obtendrá los pictogramas. Como esta acción ejecuta una llamada a una API externa, tiene un carácter asíncrono y actúa como middleware entre acciones. En la acción se ejecutará un `fetch` a la url de ARASAAC. Si todo va bien se obtendrán los datos y se ejecutará otra acción llamada *FetchPictogramsSuccess* que actualizará el Store con los datos obtenidos y automáticamente actualizará la interfaz con los resultados. Si se ha producido cualquier error de conexión se ejecutará

una acción llamada *FetchPictogramsFailure* donde se almacenará el error y se comunicará en la vista.

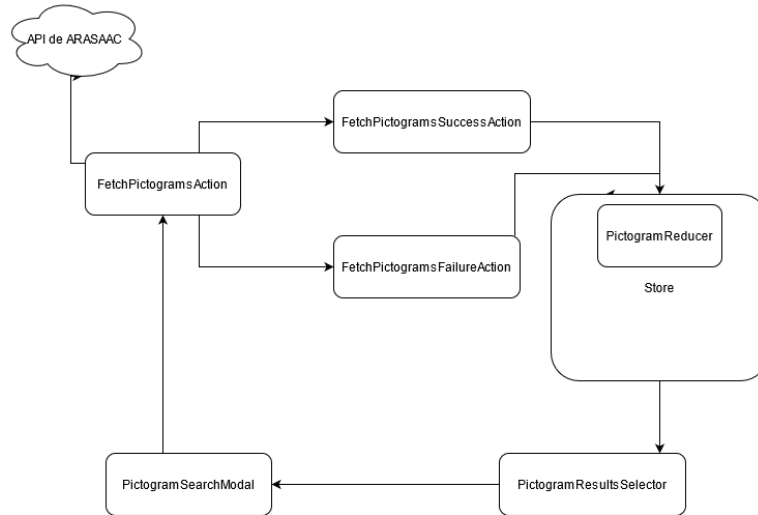


Figura 6.31: Diagrama de clases del buscador de pictogramas

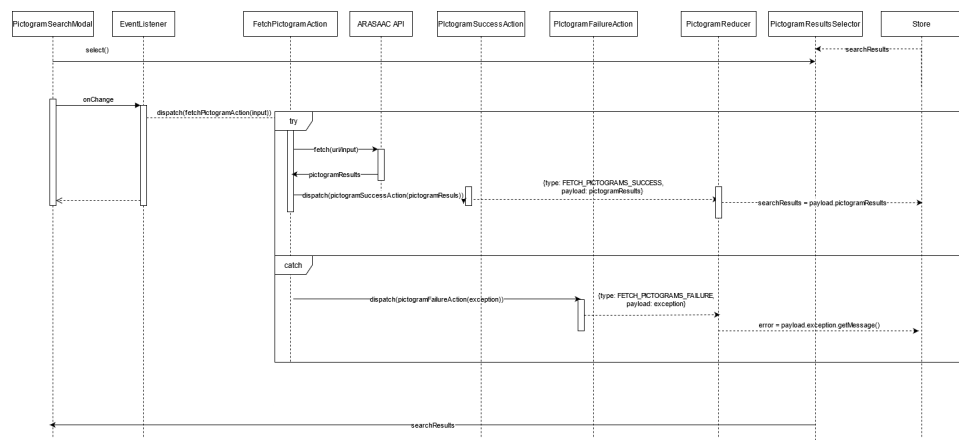


Figura 6.32: Diagrama de secuencia del buscador de pictogramas

6.3.2.4. Sopa de letras

En la Figura 6.21 se puede ver el aspecto de la aplicación al seleccionar sopa de letras. La creación del ejercicio de sopa de letras solicita los siguientes parámetros:

- Filas: El número de filas que tendrá la sopa de letras.
- Columnas: El número de columnas que tendrá la sopa de letras.
- Palabras: Las palabras que se insertarán en la sopa de letras.
- Dirección: Las direcciones en las que se quiere insertar la palabra (vertical, horizontal y/o diagonal). Como mínimo se debe activar una dirección para poder generar la sopa de letras.
- Palabras máximas: El número de palabras máximas que se insertarán en la sopa de letras.
- Activar escritura al revés: Activa la probabilidad de escribir cada palabra al revés. En el caso de que esta opción no se active, la probabilidad será cero. Si se activa, aparecerá una barra deslizante para establecer dicha probabilidad.

Los parámetros obligatorios para poder generar la sopa de letras son las filas, las columnas, las palabras y la/s dirección/es. Así mismo, existe otra opción de carácter opcional y es la posibilidad de mostrar en el editor las palabras a buscar junto al enunciado.

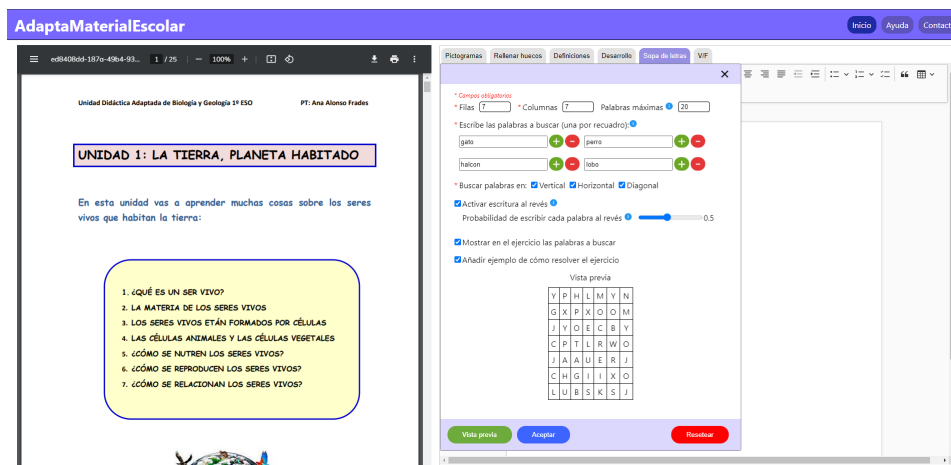


Figura 6.33: Vista previa de la sopa de letras

Una vez que se han rellenado los campos obligatorios se activarán los botones de Vista previa y Aceptar. El botón Vista previa genera primero la sopa de letras y la muestra al usuario antes de escribirla en el editor (en la Figura 6.33 se muestra la vista previa de la sopa de letras generada). En cambio, el botón Aceptar genera la sopa de letras y la escribe directamente en el editor, sin mostrarla antes al usuario. En ambos casos, si ocurre algún

error (por ejemplo, que el número de filas y columnas sea igual o menor que cero) se mostrará al usuario el mensaje de error correspondiente.

En la Figura 6.35 se muestra el diagrama de secuencia de esta funcionalidad. Tal y como puede verse en dicho diagrama, al pulsar Aceptar ocurre lo siguiente:

1. Se hace una llamada al despachador pasándole como parámetro *updateForceReadyToPreview(false)*, para ocultar la vista previa en caso de que esté visible.
2. Hay que distinguir dos casos fundamentales:
 - Si ya hay una sopa de letras creada, es decir, el objeto *WordSearchObject* es distinto de null, significa que anteriormente se ha pulsado el botón de Vista Previa, o el botón Aceptar pero hay un error. En este caso, si hay un mensaje de error se hace una llamada a *generateWordSearch()*. Si no hay error se procederá a escribir en el editor la sopa de letras generada a través de la función *writeInEditor()* (fase 4).
 - Si no hay una sopa de letras creada, es decir, el objeto *WordSearchObject* es null, llamaremos directamente a *generateWordSearch()*.
3. La función *generateWordSearch()* consta de dos fases:
 - a) Se llama al despachador pasándole como parámetro *createWordSearch()*. El despachador buscará en *Actions* el tipo de acción que es (en este caso, *CREATE_WORDSEARCH*). Posteriormente llamará al reductor a través de *wordSearchRedutor()*, pasándole como parámetros el estado (*state*) del componente y la acción buscada anteriormente. A continuación, buscará en el *switch* el tipo de acción y se procederá a crear la sopa de letras llamando a *createWordSearch()* pasándole como parámetros el objeto *wordSearchObject* y el objeto de configuración de la sopa de letras, *options*. Esta función devolverá una sopa de letras en caso de que las filas y columnas sean mayores que cero. En caso contrario, o si ha habido alguna excepción, devolverá null. En la Figura 6.34 se puede ver un diagrama con el flujo de acción, reductor, etc., concretos para este caso.
 - b) Se llama al despachador pasándole como acción *updateError()*. El despachador buscará en *Actions* el tipo de acción que es (que en este caso es *UPDATE_WORDSEARCH_ERROR*). Después llamará al reductor donde buscará el tipo de acción en el *switch* y

actualizará el mensaje de error llamando a la función *manageError()* pasándole como parámetros el error, el objeto *wordSearchObject* y la longitud del array *dictionary*. El error solo puede tomar tres estados:

- Si el objeto *wordSearchObject* es null, el error será “Las filas y columnas deben tener un valor positivo mayor que 0”.
 - Si el objeto *wordSearchObject* es distinto a null:
 - Hay que ver si la longitud de words (una de las funciones de la sopa de letras, ver sección 4.6) es menor que la longitud del array *dictionary*. En este caso, el error será “No todas las palabras introducidas están en la sopa de letras. Prueba a cambiar el valor de las filas, columnas y/o número máximo de palabras”.
 - En caso contrario, no habrá mensaje de error.
4. Por último, se hará una llamada a *writeInEditor()*, una función local de *WordSearchModal*, cuyo objetivo es escribir en el editor la sopa de letras. Para que se escriba en el editor, se ejecuta una función del editor llamada *execute()* donde se le pasa como parámetros el nombre del comando (*insertWordSearch*) y un objeto de datos con la sopa de letras, las palabras que se han insertado en la sopa de letras, la opción de mostrar las palabras a buscar en la sopa de letras y la opción de añadir un ejemplo de cómo resolver el ejercicio.

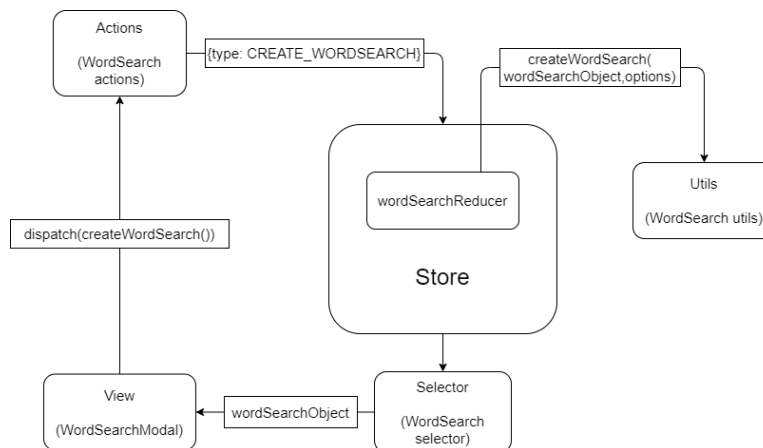


Figura 6.34: Diagrama de flujo de la generación de la sopa de letras

En la Figura 6.36 se muestra, en la parte superior, el resultado de la generación de la sopa de letras si se usa como configuración los datos de la Figura 6.33, y en la parte inferior, el resultado de la generación de la sopa de letras si se desactivan las opciones de mostrar en el ejercicio las palabras

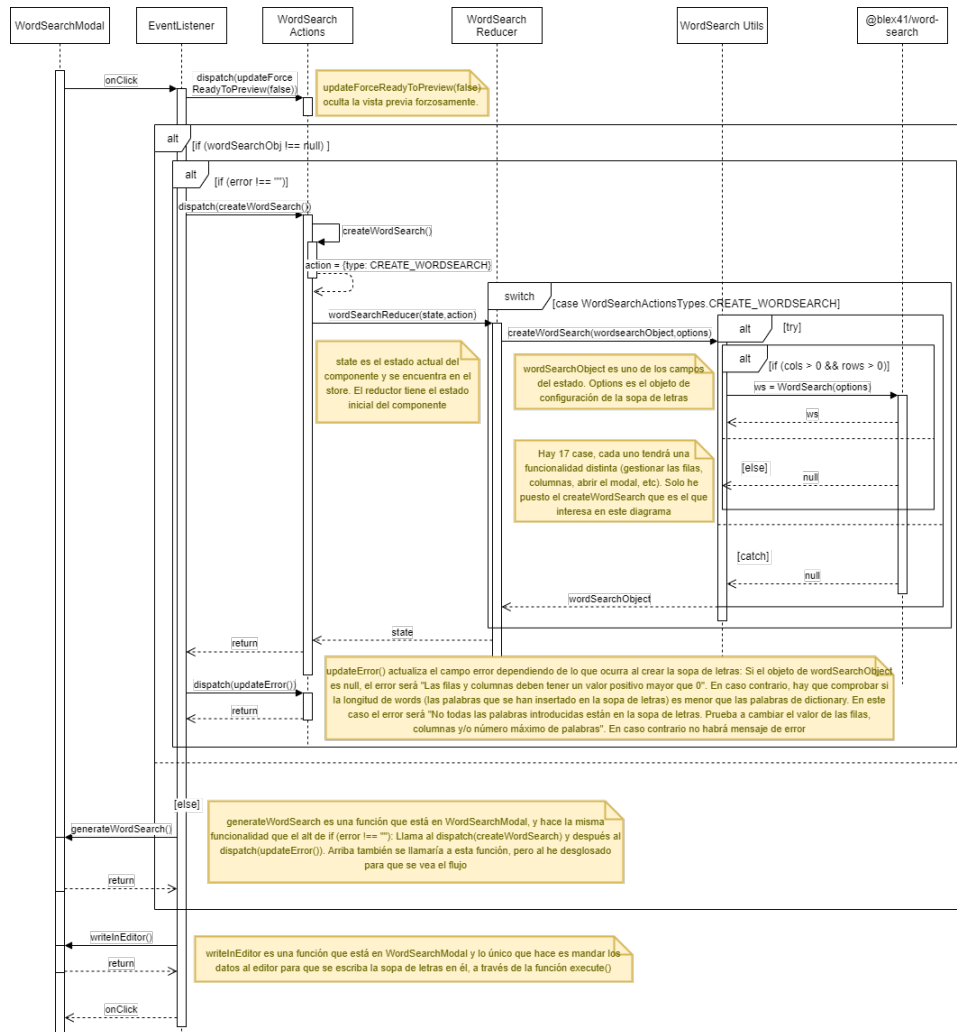


Figura 6.35: Diagrama de secuencia de la generación de la sopa de letras

a buscar, añadir ejemplo de cómo resolver el ejercicio y activar escritura al revés, y usando los mismos parámetros de la Figura 6.33.

6.3.2.5. Definiciones

Al seleccionar esta opción en nuestra aplicación se muestra la interfaz de la Figura 6.19. Para crear este tipo de ejercicio, hay que rellenar los siguientes campos:

- Conceptos a definir: contiene los conceptos a definir. Cada concepto

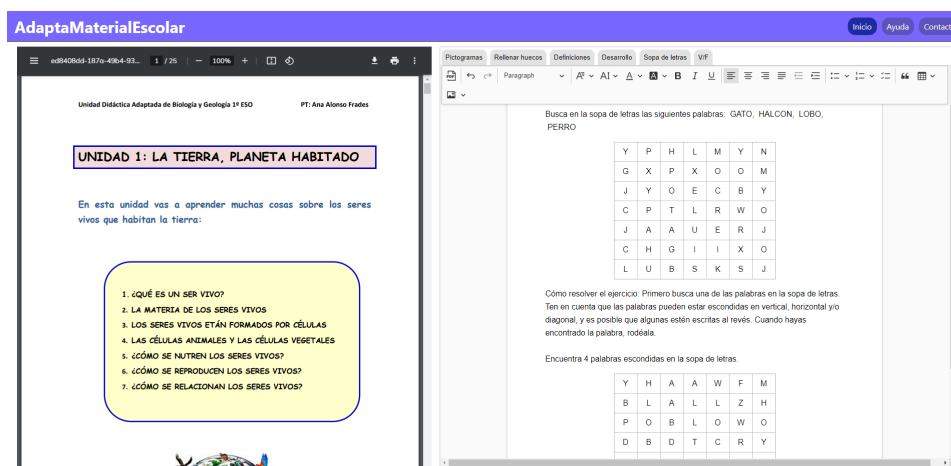


Figura 6.36: Resultado de la generación de la sopa de letras

debe ir en un input de texto. Dentro de éstos, si se pulsa a la tecla Enter, se podrán introducir más conceptos. También se puede hacer clic en el botón “más” para añadir más inputs. Para quitar alguna fila, se tendrá que hacer clic en el botón “menos”. Ambos botones están ubicados en la derecha de cada input.

- Número de líneas: número máximo de líneas para definir el concepto. Por defecto es 1.
- La forma en la que se muestran las definiciones: puede mostrarse como una lista no ordenada o una lista ordenada. Por defecto está establecido en mostrarse como una lista no ordenada.
- Añadir espacio extra entre líneas: añade un espacio más grande entre líneas para los estudiantes que tengan una letra más grande.

En la Figura 6.37 se muestra un diagrama con el flujo de acción, reductor, etc., cuando se modifica el texto de una de las definiciones y en la Figura 6.38, un diagrama de secuencia de la misma funcionalidad. Tal y como se puede ver en el diagrama de secuencia ocurre lo siguiente:

1. Se llama al despachador pasándole como parámetro *updateDefinitions-Text(e.target.value, i)*, donde *e.target.value* es el valor del campo del input que lo ha llamado (el texto introducido), e *i* la posición en el que se encuentra en el array de definiciones.
2. El despachador buscará en *Actions* el tipo de acción que es (*UPDATE_DEFINITIONS_TEXT*) junto con su *payload* (el texto introducido y la posición). Posteriormente llamará al reductor a través de

definitionsReducer() pasándole como parámetros el estado (*state*) del componente y la acción devuelta anteriormente. A continuación buscará en el *switch* el tipo de acción y procederá a actualizar el estado del componente, concretamente la variable *text*, el array donde se almacena cada definición. Esta actualización la hace la función *update()* pasándole como parámetros el array a modificar, y un objeto con el índice y el nuevo valor. Tras actualizarlo, devolverá el estado modificado.

3. Por último, tras realizar este proceso, se actualizará la interfaz con el nuevo estado devuelto por el reductor.

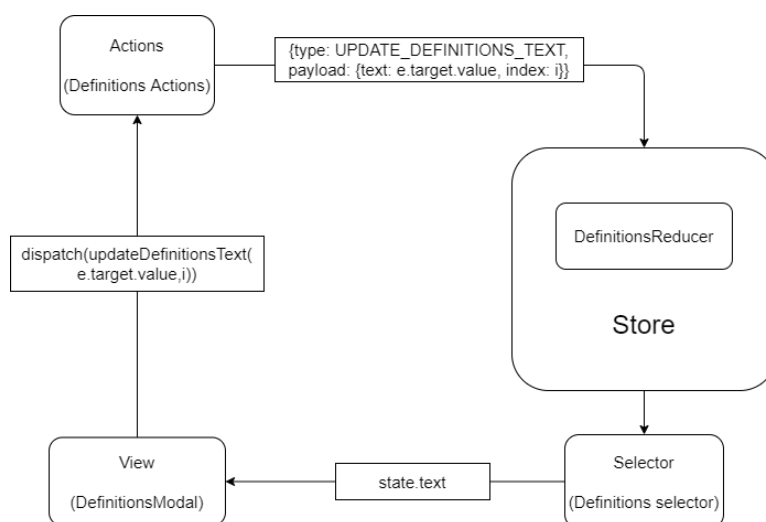


Figura 6.37: Diagrama de flujo cuando se modifica el texto de una definición

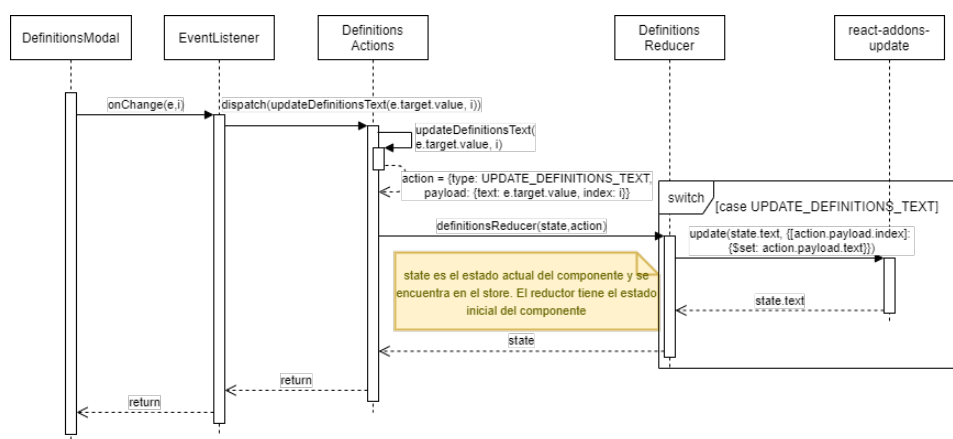


Figura 6.38: Diagrama de secuencia cuando se modifica el texto de una definición

Una vez que se han rellenado los campos obligatorios (conceptos a definir, número de líneas y la forma en la que se muestran las definiciones), el botón Aceptar se habilitará. Al hacer clic en él se procederá a escribir en el editor cada concepto, seguido del número de líneas especificado por el usuario. Para ello, se llama a la función *execute()* del editor, pasándole como parámetros el nombre del comando (*insertDefinitions*) y el objeto de datos con la lista de conceptos, el número de líneas, el tipo de lista y las opciones para añadir espacio extra entre líneas y para añadir un ejemplo de cómo resolver el ejercicio. En la Figura 6.40 se muestra el resultado de este ejercicio al usar las opciones de la Figura 6.39 en la parte superior. La parte inferior es el resultado de usar las mismas opciones pero cambiando el tipo de lista a lista no ordenada, añadiendo la opción de añadir espacio extra entre líneas y quitando la opción de añadir ejemplo de cómo resolver el ejercicio.

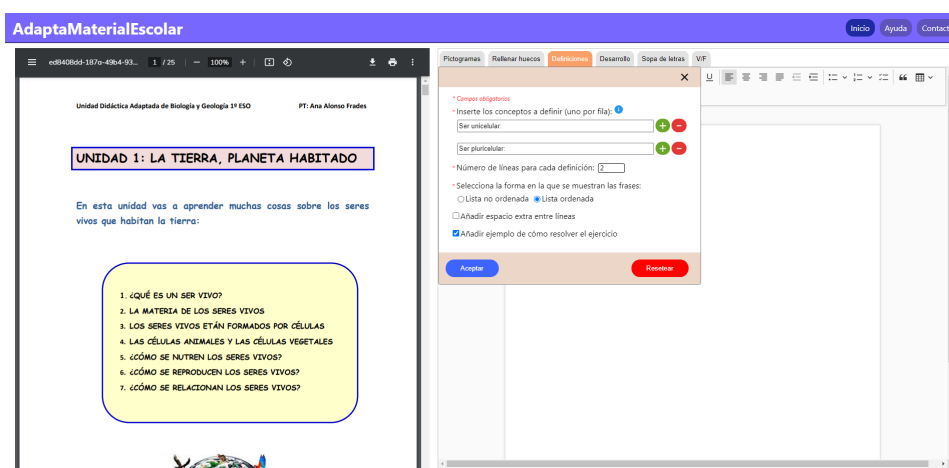


Figura 6.39: Ejemplo I de opciones del ejercicio de definiciones

6.3.2.6. Pregunta para desarrollar

La interfaz de este ejercicio se puede ver en la Figura 6.20. Para crear el ejercicio, hace falta escribir el enunciado y elegir el número de líneas máximo para desarrollarlo. Así mismo, se ha incluido una opción no obligatoria para añadir espacio extra entre líneas en caso de que algún estudiante tenga un tamaño de letra mayor.

En la Figura 6.41 se muestra un diagrama con el flujo de acción, reductor, etc., cuando se modifica el enunciado de este ejercicio y en la Figura 6.42, un diagrama de secuencia de la misma funcionalidad. Tal y como se puede ver en el diagrama de secuencia ocurre lo siguiente:

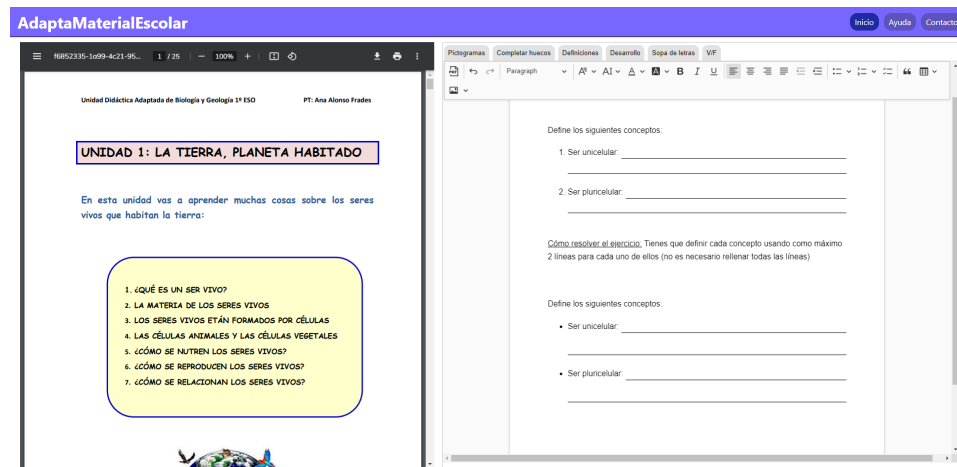


Figura 6.40: Resultado del ejercicio de definiciones

1. Se llama al despachador pasándole como parámetro *updateDevelopText(e.target.value)*, donde *e.target.value* es el valor del campo del input que lo ha llamado (el enunciado modificado).
2. El despachador buscará en *Actions* el tipo de acción que es (*UPDATE_DEVELOP_TEXT*) junto con su *payload* (el enunciado modificado). Posteriormente llamará al reductor a través de *developReducer()* pasándole como parámetros el estado (*state*) del componente y la acción devuelta anteriormente. A continuación buscará en el *switch* el tipo de acción y procederá a actualizar el estado del componente, concretamente la variable *text*, un string donde se almacena el enunciado. Tras actualizarlo, devolverá el estado modificado.
3. Por último, tras realizar este proceso, se actualizará la interfaz con el nuevo estado devuelto por el reductor.

Una vez que se han rellenado los campos obligatorios, el botón Aceptar se habilitará. Al hacer clic en él se procederá a escribir, en el editor, el enunciado del ejercicio y en los siguientes renglones el número de líneas especificado por el usuario. Para ello, se llama a la función *execute()* del editor, pasándole como parámetros el nombre del comando (*insertDevelop*), y el objeto de datos con el enunciado, el número de líneas y las opciones para añadir espacio extra entre líneas y para añadir un ejemplo de cómo resolver el ejercicio. En la Figura 6.44 se muestra el resultado de este ejercicio. En la parte superior se muestra dicho resultado usando las opciones de la Figura 6.43. En la parte inferior, el mismo ejercicio pero quitando la opción de añadir el ejemplo de cómo resolver el ejercicio, y añadiendo espacio extra entre líneas.

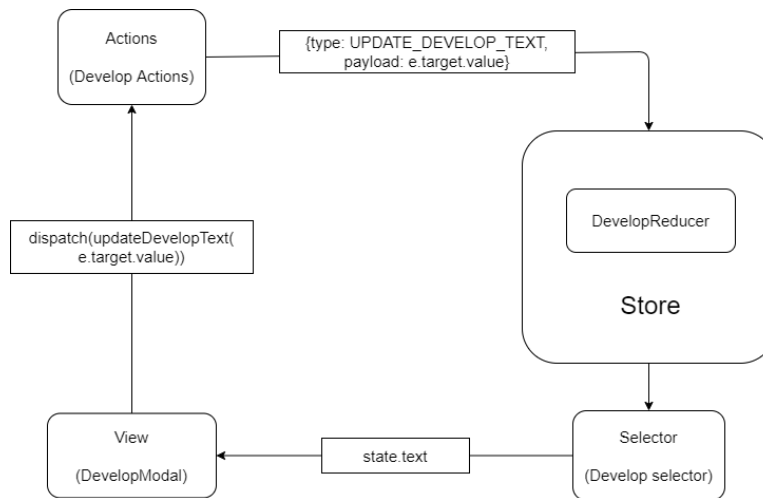


Figura 6.41: Diagrama de flujo cuando se modifica el enunciado del ejercicio pregunta para desarrollar

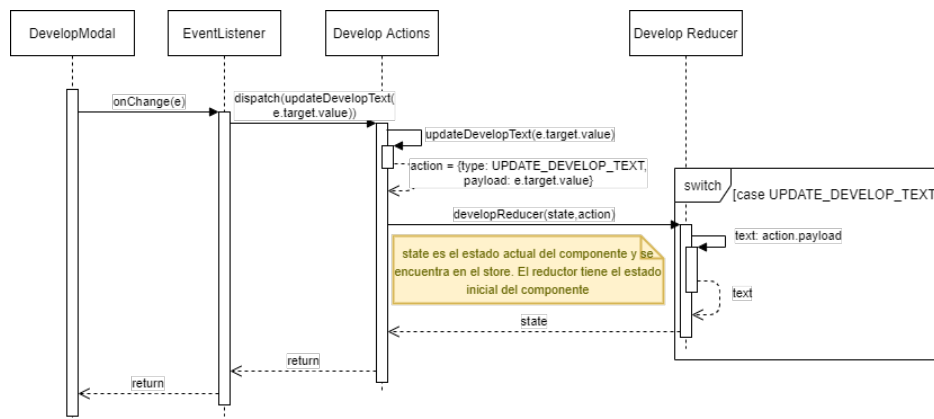


Figura 6.42: Diagrama de secuencia cuando se modifica el enunciado del ejercicio pregunta para desarrollar

6.3.2.7. Verdadero o Falso

Al seleccionar esta opción en nuestra aplicación, se muestra la interfaz de la Figura 6.22. Para crear un ejercicio de verdadero y falso se necesitan rellenar los siguientes campos:

- Frases: Contendrá las frases del ejercicio. Hay que escribir una frase en cada input de texto. Para añadir más inputs, se puede pulsar la tecla Enter dentro de uno de los inputs, o pulsar el botón “más”. Para quitarlos, se tendrá que hacer clic en el “menos”. Ambos botones se

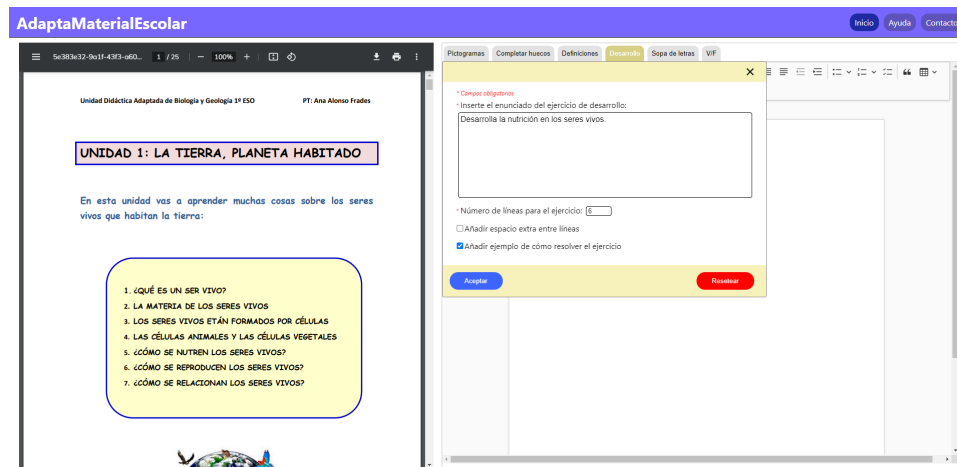


Figura 6.43: Ejemplo de opciones del ejercicio pregunta para desarrollar

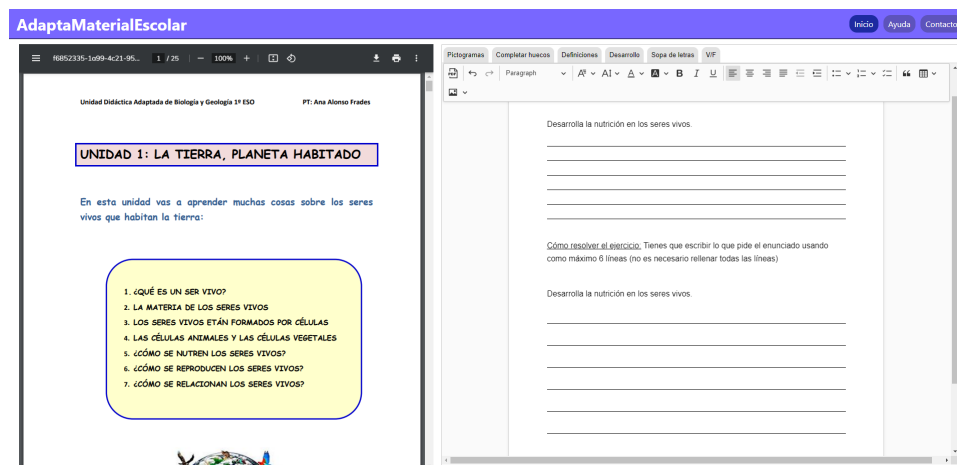


Figura 6.44: Resultado de ejercicio pregunta para desarrollar

encuentran ubicados en la derecha de cada input.

- La forma en la que se muestran las frases: puede ser como una lista no ordenada o una lista ordenada. Por defecto está seleccionada la lista no ordenada.

En la Figura 6.45 se muestra un diagrama con el flujo de acción, reductor, etc., cuando se añaden más inputs y en la Figura 6.46, un diagrama de secuencia de la misma funcionalidad. Tal y como se puede ver en el diagrama de secuencia ocurre lo siguiente:

1. Se llama al despachador pasándole como parámetro *addMoreTrueFal-*

se().

2. El despachador buscará en *Actions* el tipo de acción que es (en este caso, *ADD_MORE_TRUEFALSE*). Posteriormente llamará al reductor a través de *trueFalseReducer()* pasándole como parámetros el estado (*state*) del componente y la acción devuelta anteriormente. A continuación buscará en el *switch* el tipo de acción y procederá a actualizar el estado del componente, concretamente la variable *text*, un array que almacena cada frase de verdadero o falso. Esta actualización añade al final del array una cadena de texto vacía. Tras actualizarlo, devolverá el estado modificado.
3. Por último, tras realizar este proceso, se actualizará la interfaz con el nuevo estado devuelto por el reductor, en el que se mostrará otro input para escribir otra frase de verdadero o falso, junto con los botones de añadir más filas y quitar fila.

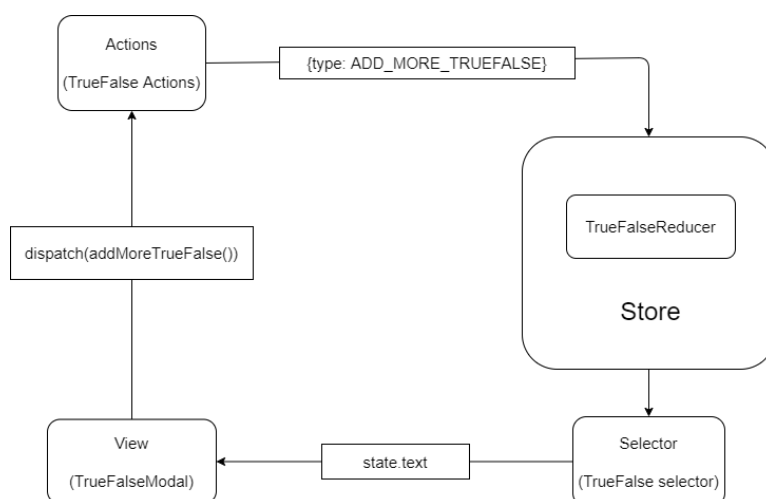


Figura 6.45: Diagrama de flujo cuando añaden más filas al ejercicio de verdadero o falso

Una vez que se han rellenado los campos obligatorios mencionados anteriormente, el botón Aceptar se habilitará. Al hacer clic en él se procederá a escribir en el editor cada frase, seguido de un recuadro para escribir V si es verdadero o F si es falso. Para ello, se llama a la función *execute()* del editor, pasándole como parámetros el nombre del comando (*insertTrueFalse*) y un objeto de datos con la lista de oraciones, el tipo de lista y la opción para añadir un ejemplo de cómo resolver el ejercicio. En la Figura 6.48 se muestra el resultado del ejercicio usando las opciones de la Figura 6.47.

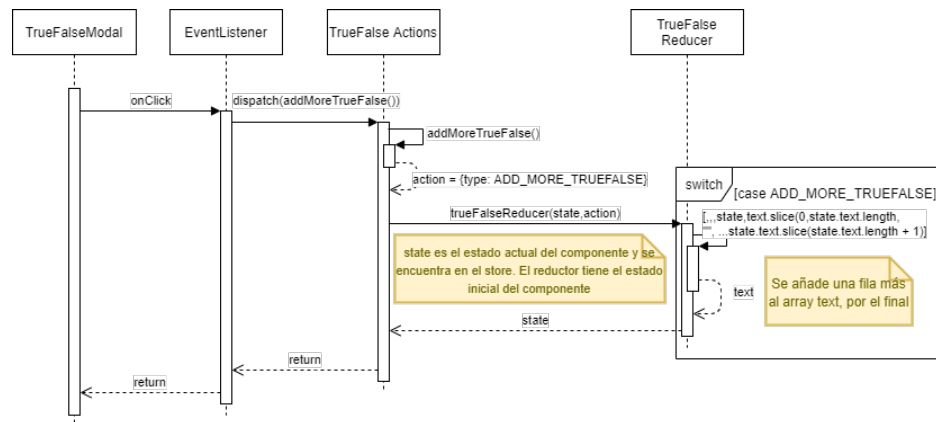


Figura 6.46: Diagrama de secuencia cuando añaden más filas al ejercicio de verdadero o falso

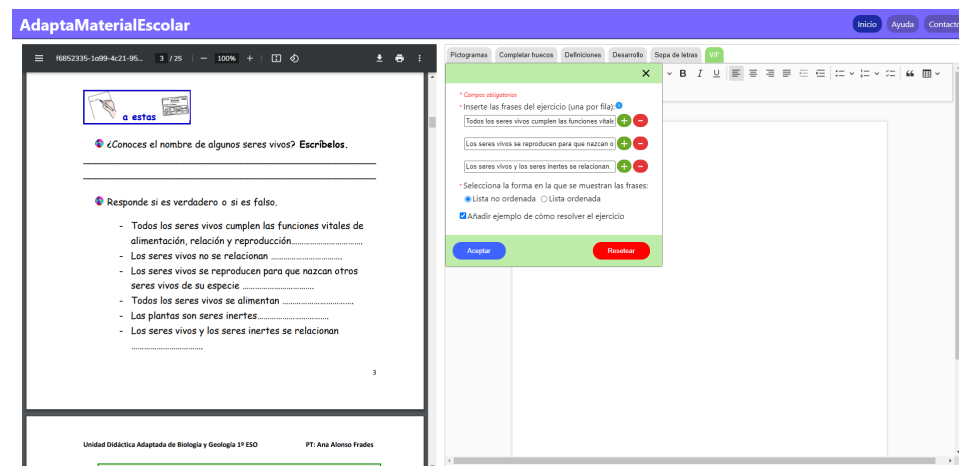


Figura 6.47: Ejemplo de opciones del ejercicio verdadero o falso

6.3.2.8. Completar huecos

El ejercicio de completar huecos tiene como finalidad escribir las palabras faltantes en los espacios en los que haya una línea. Al seleccionar esta opción en nuestra aplicación se muestra la interfaz de la Figura 6.18. Este ejercicio consta de dos fases:

- En la primera fase hay que insertar el texto del que se quiera quitar palabras. Una vez que se ha insertado, el botón Seleccionar palabras se habilitará. El texto se puede seleccionar y copiar del documento fuente y pegarlo en el *textarea*. En la Figura 6.49 se muestra un ejemplo de la

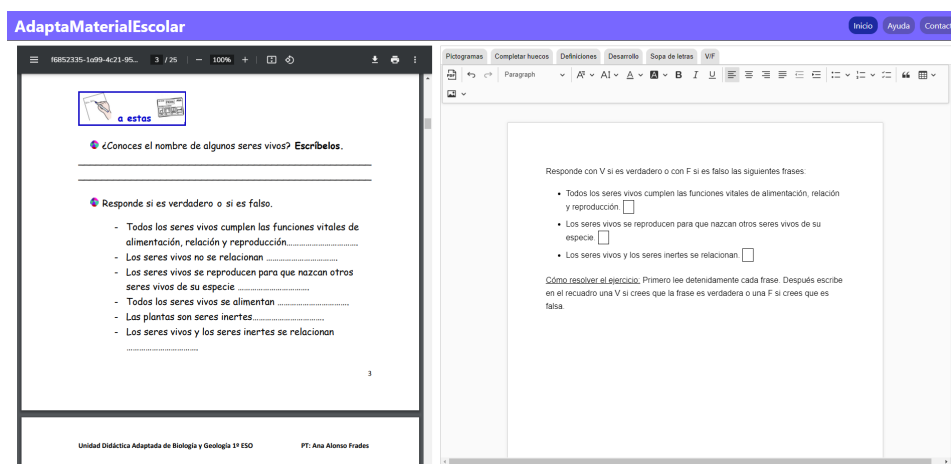


Figura 6.48: Resultado de ejercicio verdadero o falso

interfaz de edición en la que se debe insertar el texto, y en el que una vez insertado, el botón Seleccionar palabras se habilitará.

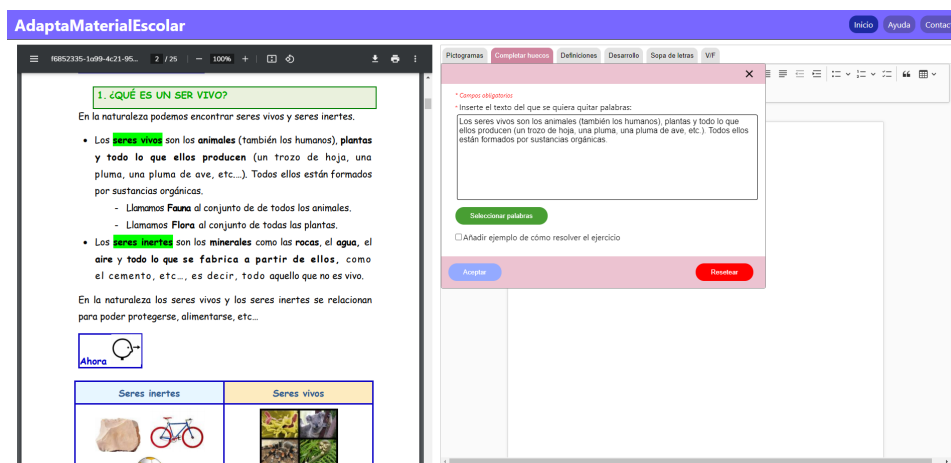


Figura 6.49: Ejemplo de edición del texto del ejercicio completar huecos

- La segunda fase se alcanza cuando se hace clic en el botón de Seleccionar palabras. Para quitar una palabra basta con hacer clic en ella. Si se quiere volver a ver la palabra, hay que volver a hacer clic en ella. Para volver a editar el texto (volver a la fase uno), simplemente hay que hacer clic en el botón Editar texto. En la Figura 6.50 se puede ver la interfaz de selección de las palabras. En ésta, se han seleccionado las palabras vivos, animales, plantas y orgánicas.

En la Figura 6.51 se muestra un diagrama de secuencia cuando se hace

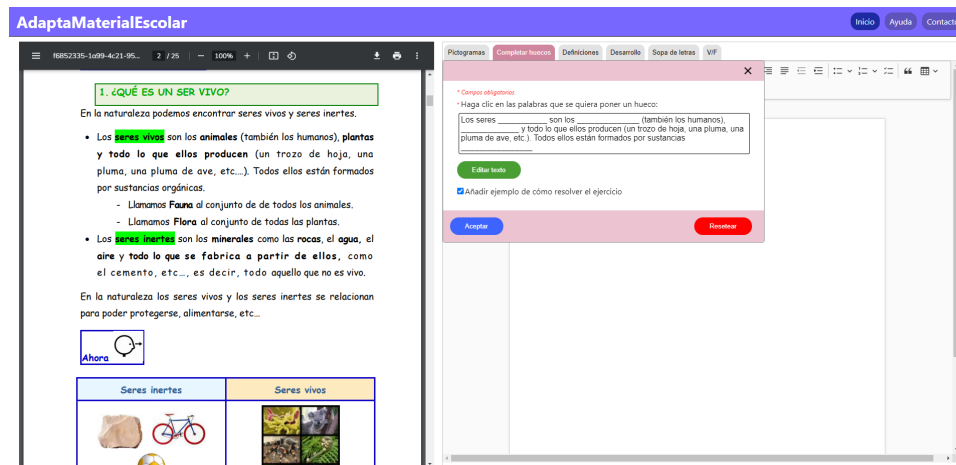


Figura 6.50: Ejemplo de selección de palabras del ejercicio completar huecos

clic en el botón Seleccionar palabras. Tal y como se puede ver en el diagrama de secuencia ocurre lo siguiente:

1. Se hace una llamada a la función *changeToSelection()*. Esta función consta de dos partes:
 - a) Se hace una llamada al despachador pasándole como parámetro *updateMode(mode)*, donde *mode* puede ser o *edition* o *selection*. En este caso es *selection*. El despachador buscará en *Actions* el tipo de acción que es (*UPDATE_FILLGAPS_MODE*), junto con su *payload*. Después llamará al reductor a través de *fillGapsReducer()* pasándole como parámetros el estado (*state*) del componente y la acción devuelta anteriormente, buscará el tipo de acción en el *switch* y actualizará el modo a *selection*. Después devolverá el estado actualizado.
 - b) Se hace una llamada al despachador pasándole como parámetro *updateTextSelection()*. El despachador buscará en *Actions* el tipo de acción que es (*UPDATE_FILLGAPS_TEXTSELECTION*). Posteriormente llamará al reductor a través de *fillGapsReducer()* pasándole como parámetros el estado (*state*) del componente y la acción devuelta anteriormente, buscará el tipo de acción en el *switch* y actualizará la variable *textSelection*. Esta variable contiene las palabras del texto separadas por espacios. Después devolverá el estado actualizado. Una vez finalizado este proceso, se cambiará a la interfaz de selección de palabras (Figura 6.50).
2. Al seleccionar una palabra se llama a la función *handleOnClick(e,i)* pasándole como parámetros el evento que lo ha llamado (*e*) y la po-

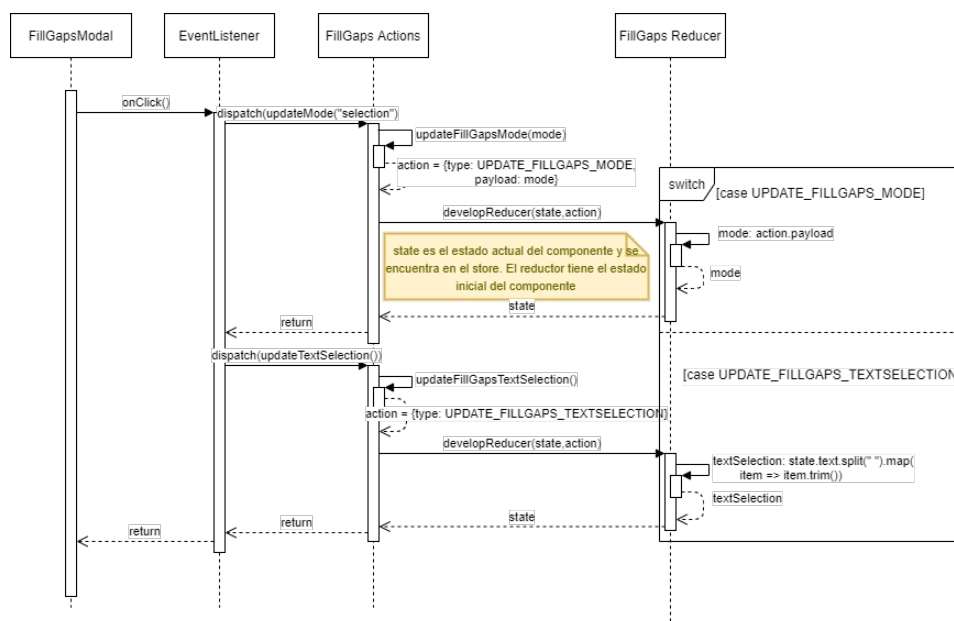


Figura 6.51: Diagrama de secuencia cuando se hace clic sobre Seleccionar palabras en el ejercicio completar huecos

sición en el array de *textSelection* (i). En la Figura 6.53 se muestra un diagrama de secuencia cuando se hace clic en una palabra, y en la Figura 6.52, un diagrama de flujo de esta funcionalidad. Al seleccionar una palabra pueden ocurrir dos situaciones:

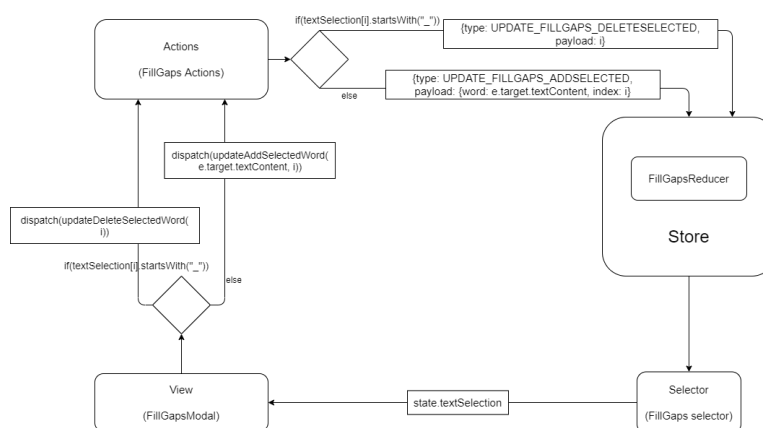


Figura 6.52: Diagrama de flujo al seleccionar una palabra del ejercicio completar huecos

- a) Si la palabra comienza con una guión bajo, se llama al despachador pasándole como parámetro *updateDeleteSelectedWord(i)*,

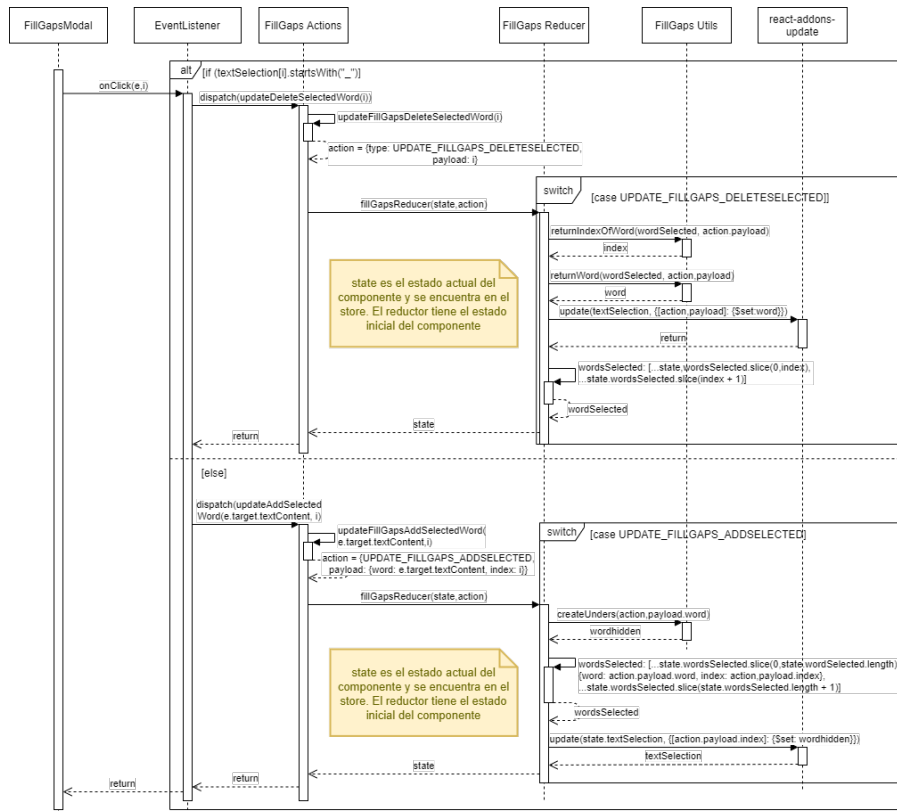


Figura 6.53: Diagrama de secuencia cuando se hace clic en una palabra en el ejercicio completar huecos

donde i es la posición de la palabra en el array *textSelection*. El despachador buscará en *Actions* el tipo de acción que es (*UPDATE_FILLGAPS_DELETESELECTED*) junto con su *payload* (i). Posteriormente se llamará al reducer a través de *fillGapsReducer()* pasándole como parámetros el estado (*state*) del componente y la acción, buscará el tipo de acción en el *switch* y procederá a actualizar dos variables: *textSelection*, donde se recupera la palabra que se ha ocultado, y *wordSelected*, donde se busca el índice de la palabra que se ha ocultado en el array *wordsSelected*, y se elimina la palabra de éste. Después devolverá el estado modificado.

- b) Si la palabra no comienza con guión bajo, se llama al despachador pasándole como parámetro *updateAddSelectedWord(textContent, i)*, donde *textContent* es la palabra e i es la posición en el array *textContent*. El despachador buscará en *Actions* el tipo de acción que es (*UPDATE_FILLGAPS_ADDSELECTED*) junto con su *payload* (i). Posteriormente se llamará al reducer a través de *fill-*

GapsReducer() pasándole como parámetros el estado (*state*) del componente y la acción, buscará el tipo de acción en el *switch* y procederá a actualizar dos variables: *textSelection*, donde se sustituye la palabra por guiones bajos, y *wordsSelected*, donde se añade un objeto con la palabra y la posición en el array *textSelection* para poder recuperarla después. Por último devolverá el estado modificado.

Una vez que se ha seleccionado al menos una palabra, el botón Aceptar se habilitará. Al hacer clic en éste se escribirá el texto modificado en el editor. Para ello, se llamará a la función *execute()* del editor, pasándole como parámetros el nombre del comando (*insertFillGaps*) y el objeto de datos con el texto modificado, las palabras escondidas y la opción para añadir un ejemplo de cómo resolver el ejercicio. En la Figura 6.54 se ve el resultado del ejercicio al usar las opciones de las Figuras 6.49 y 6.50.

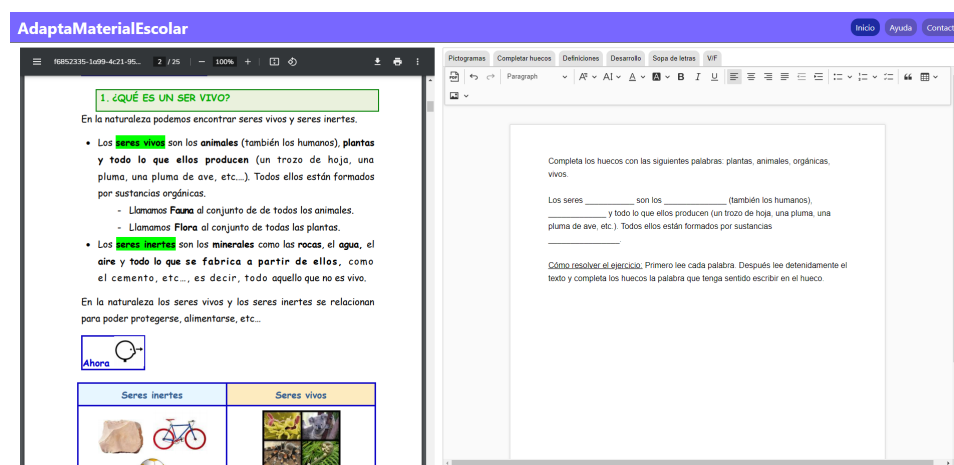


Figura 6.54: Resultado del ejercicio de completar huecos

6.3.3. Estructura del código

La organización de directorios del proyecto gira en torno al patrón de diseño Redux. El código de la funcionalidad de la aplicación se estructura principalmente en cuatro carpetas: *components*, *pages*, *ckeditor* y *redux*. Además, contamos con el fichero *App*, que supone el punto de inicio del programa. El contenido de cada carpeta es el siguiente:

- **Components:** Representado en la Figura 6.55, contiene cada componente que representa una funcionalidad para la aplicación, o parte de

ella, y todos cuentan, como mínimo, con los siguientes ficheros:

- Fichero modal de la funcionalidad del componente.
- Fichero que realiza las modificaciones pertinentes para ejecutar la funcionalidad.
- Fichero de estilo del componente.

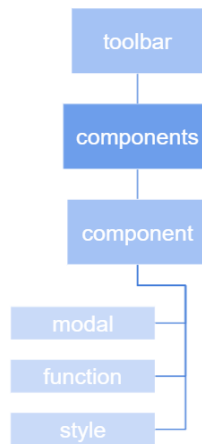


Figura 6.55: Ejemplo de organización de Components

Uno de los componentes más importantes es el de la barra de herramientas (*Toolbar*), que sirve como hilo conductor de la selección de los diferentes tipos de adaptaciones que pueden agregarse al documento, mediante el patrón de diseño del despachador (*dispatcher*).

- **Pages:** Representado en la Figura 6.56, contiene la página del editor, que permite cargar un fichero inicial para comenzar la adaptación, y la página de ayuda, que muestra información relevante sobre la aplicación y su uso. En los dos casos se cuenta con:
 - Fichero de funcionalidad de la página.
 - Fichero de estilo de la página.
- **Redux:** Representado en la Figura 6.57, contiene la implementación del patrón de diseño que se ha aplicado a toda la aplicación, basado en el almacenamiento de los datos que necesita cada componente. La carpeta consta del Store y el reducer, elementos necesarios para el desarrollo del mismo. Además, se incluye también cada componente principal con los siguientes ficheros:



Figura 6.56: Ejemplo de organización de Pages

- Fichero action: controla las diferentes acciones que puede ejecutar cada componente.
- Fichero reducer: controla el estado de los componentes según el tipo de acción que realicen, partiendo de un estado inicial.
- Fichero selectors: devuelve el valor de los atributos de cada componente.
- Fichero types: contiene un listado de acciones posibles para cada componente.
- Fichero utils: contiene funciones auxiliares para actualizar el valor de algunos parámetros del estado del componente. No todos los componentes tendrán este fichero.

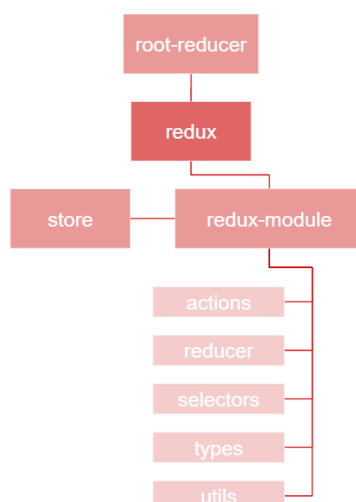


Figura 6.57: Ejemplo de organización de Redux

- **CKEditor:** Representado en la Figura 6.58, contiene todos componentes software o plugins que se han desarrollado para la aplicación, permitiendo la interacción a partir de comandos que controlan las diferentes funcionalidades. Consta de un fichero que devuelve la instancia del editor, CKEditor 5, y una carpeta para cada plugin, con los siguientes ficheros:

- Fichero de implementación del plugin.
- Fichero de inserción del plugin mediante el patrón de diseño de comando (*command*).

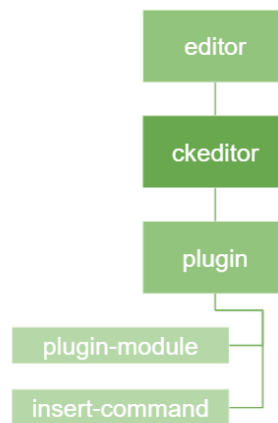


Figura 6.58: Ejemplo de organización de CKEditor

Capítulo 7

Evaluación de AdaptaMaterialEscolar

RESUMEN: En este capítulo se explicará cómo se realizó la evaluación del proyecto AdaptaMaterialEscolar con los usuarios finales y los resultados obtenidos.

7.1. Introducción

Una vez terminada la primera versión de la aplicación web AdaptaMaterialEscolar, organizamos la evaluación que necesitábamos para cumplir la intención principal de ésta, la de confirmar si finalmente había resultado útil nuestro proyecto para los docentes que tuvieran que realizar algún tipo de adaptación curricular al alumnado que lo necesitara, durante cualquier etapa académica.

Con el fin de hacer una valoración detallada, los aspectos que queríamos evaluar en nuestro proyecto eran: el diseño de la interfaz, la usabilidad del sistema al completo, cada funcionalidad independiente y las posibles mejoras que se podrían implementar en el futuro.

7.2. Diseño de la evaluación

Para que los usuarios finales pudieran evaluar la aplicación, redactamos un examen de prueba, a partir de un tema adaptado de Ciencias Naturales que nos facilitaron las propias profesoras del Aula TEA, con cada una de las funcionalidades que estaban desarrolladas para el momento de la prueba, que se encuentra en el Anexo A, con la intención de replicarlo con ellos durante la evaluación y, además, se les facilitó el enlace a la aplicación web para que después pudieran practicar libremente. Facilitamos la realización de la evaluación y análisis de los resultados de nuestra aplicación web desarrollando una encuesta en Google Forms para los usuarios finales. Las preguntas del cuestionario se estructuraron según el ámbito y fueron las siguientes:

- **Preguntas generales.** Datos demográficos del usuario.
 - ¿Cuál es tu género?
 - ¿Eres docente?
- **Preguntas para docentes.** Recogen el nivel académico en el que imparten clase y el centro educativo en el que trabajan.
 - ¿Podrías decirnos en qué nivel del sistema educativo eres docente?
 - ¿Podrías decirnos de qué centro educativo? (Pregunta no obligatoria).
- **Preguntas sobre usabilidad.** Siguen el cuestionario de la Escala de Usabilidad de un Sistema (Devin, 2017), o SUS (System Usability Scale) por sus siglas en inglés. SUS es uno de los cuestionarios para usabilidad más utilizados. Las preguntas de este cuestionario son las siguientes:
 - Creo que me gustaría utilizar este sistema con frecuencia.
 - El sistema me parece innecesariamente complejo.
 - Me pareció que el sistema era fácil de utilizar.
 - Creo que necesitaría el apoyo de un técnico para ser capaz de utilizar este sistema.
 - Me parece que las distintas funciones de este sistema estaban bien integradas.
 - Me pareció que había demasiada inconsistencia en este sistema.
 - Me imagino que la mayoría de la gente aprendería a utilizar este sistema muy rápidamente.
 - El sistema me pareció muy engorroso de usar.

- Me sentí muy seguro al utilizar el sistema.
- Tuve que aprender muchas cosas antes de poder ponerme en marcha con este sistema.

A partir de las respuestas a estas preguntas se obtiene una puntuación de la siguiente manera:

- En las preguntas pares se debe restar a 5 el valor que asigne el usuario.
- En las preguntas impares se debe restar 1 al valor que asigne el usuario.
- El resultado de cada pregunta (tras realizar las operaciones anteriores) debe sumarse y multiplicarse finalmente por 2.5.

Si el resultado que se obtiene es cercano a 100, que es la máxima puntuación que se podría obtener con las preguntas del cuestionario, significa que la usabilidad del proyecto es buena para el usuario. En caso contrario, habría que realizar cambios para mejorar y facilitar su uso.

- **Preguntas sobre diseño.** Obtienen información sobre la estética de la aplicación.
 - ¿Te parece que la aplicación es visualmente atractiva?
 - ¿Te gustaría que se realizaran cambios en la estética de la aplicación? (Pregunta no obligatoria).
- **Preguntas sobre funcionalidades.** Obtienen información sobre la dificultad de uso de cada funcionalidad y recopilar nuevas adaptaciones o mejoras que se desean para la aplicación.
 - Selecciona la dificultad de uso de cada funcionalidad de la aplicación (1 es Muy difícil y 5 Muy fácil)
 - ¿El tamaño de la letra y el espacio es adecuado?
 - ¿El tamaño de los pictogramas es adecuado?
 - ¿Te gustaría que se añadiera alguna configuración de formato más? Por ejemplo, otros tipos de fuente, tamaño, etc. (Pregunta no obligatoria).
- **Preguntas sobre utilidad real.** Comprueban si realmente la aplicación ha cumplido con su finalidad y evaluar si debería seguir en desarrollo en el futuro.
 - ¿Te parece que la aplicación es útil?
 - ¿Crees que ayuda a reducir el tiempo de trabajo del profesorado?

- ¿Te gustaría que la aplicación siguiera en desarrollo para mejorarla?
- ¿Qué mejorarías en la aplicación? (Pregunta no obligatoria).

En las preguntas donde se debía establecer como respuesta una puntuación utilizamos una escala Likert de 5 puntos, donde las puntuaciones iban del 1 al 5, significando la valoración más baja que el usuario está “Muy en desacuerdo” con la afirmación de la pregunta y, en la más alta, “Muy de acuerdo”. Todos los apartados de la encuesta, a excepción del de usabilidad, fueron planificados por nosotros específicamente para obtener información que considerábamos relevante para el proyecto.

El cuestionario se encuentra en la dirección web: https://docs.google.com/forms/d/e/1FAIpQLSfZKu_D1C2T3UMi7gQf1qzV7Ne0MbsNigVGhc0kaZ5xez7C0w/viewform?usp=sf_link

7.3. Desarrollo de la evaluación

El viernes 28 de mayo planificamos una reunión presencial en el IES Maestro Juan de Ávila, de Ciudad Real, con las profesoras del Aula TEA, quienes han colaborado con nosotros durante todo el proyecto. Una vez allí, la orientadora del centro, María José Morales Rubio, se unió a la demostración que realizamos por su propia iniciativa e interés en nuestro proyecto. Realizamos una prueba de aproximadamente una hora de duración, en la que se les mostró cómo utilizar AdaptaMaterialEscolar y replicamos con ellas el examen que habíamos preparado.

Ese mismo día se realizó la misma demostración de la aplicación, también de forma presencial, con otra profesora del mismo instituto, Esther Valiente Moreno, que imparte las asignaturas de Emprendimiento Empresarial y Economía en el centro y cuenta con un alumno TEA en una de las clases. Por falta de tiempo no se pudo mostrar el proyecto a más profesores ni se recogieron más evaluaciones, aunque contactaron con nosotros tres docentes más que se habían interesado en AdaptaMaterialEscolar y tenían intención de aprender a utilizarlo para el curso siguiente. Uno de ellos imparte Biología y los otros dos, Historia y Geografía.

El día 1 de junio, la Jefa de Estudios y profesora del IES Pedro Álvarez de Sotomayor, de Manzanares, María de las Cruces Valiente Moreno, que imparte la asignatura de Tecnología Industrial, también contactó con

nosotros para conocer el proyecto, debido a que se interesó por los comentarios que le llegaron del IES Maestro Juan de Ávila. Nos comentó que en su centro había un alumno con sordera total a quien debían adaptar el material y organizamos una reunión por videollamada para realizar la misma demostración que al resto de profesores, para mostrarle el funcionamiento de AdaptaMaterialEscolar.

Durante las reuniones con todos los profesores de los distintos centros educativos, surgieron requisitos nuevos que se pretenden plantear como trabajo futuro en caso de que AdaptaMaterialEscolar pudiera seguir en desarrollo en los próximos años. Incluso se nos llegó a ofrecer la oportunidad de realizar un curso de formación a nuevos profesores y alumnos de máster el curso que viene, en el IES Maestro Juan de Ávila, para presentarlo como herramienta de adaptabilidad para el profesorado, independientemente de si llegábamos a cumplir con los nuevos requisitos o si manteníamos el proyecto tal y como lo vieron.

7.4. Resultado de la evaluación

Finalmente, el número de usuarios finales que participaron en la evaluación de la aplicación web y contestaron la encuesta fueron 5, todas mujeres y docentes de Educación Secundaria y/o Bachillerato, aunque de haber dispuesto de más tiempo habríamos llegado a tener 9 en total, por el interés de otros profesores. Los centros de estudios participantes fueron el IES Maestro Juan de Ávila, de Ciudad Real, y el IES Pedro Álvarez de Sotomayor, de Manzanares.

El resultado a las preguntas de usabilidad del SUS fue realmente positivo, ya que obtuvimos una puntuación final de 99, siendo 100 la máxima puntuación, por lo que se puede concluir que el sistema es fácilmente manejable por los usuarios con una formación mínima. La gráfica con la media de las puntuaciones de cada pregunta se puede ver en la Figura 7.1 y los cálculos del resultado en la Tabla 7.1.

Pregunta 1	$5 - 1 = 4$
Pregunta 2	$5 - 1,2 = 3,8$
Pregunta 3	$5 - 1 = 4$
Pregunta 4	$5 - 1 = 4$
Pregunta 5	$5 - 1 = 4$
Pregunta 6	$5 - 1 = 4$
Pregunta 7	$5 - 1 = 4$

Tabla 7.1 continuada de la página anterior

Pregunta 8	$5 - 1 = 4$
Pregunta 9	$4,8 - 1 = 3,8$
Pregunta 10	$5 - 1 = 4$
Resultado	$((4 \cdot 8) + (3,8 \cdot 2) \cdot 2,5) = 99$

Tabla 7.1: Cálculo del resultado del cuestionario de SUS

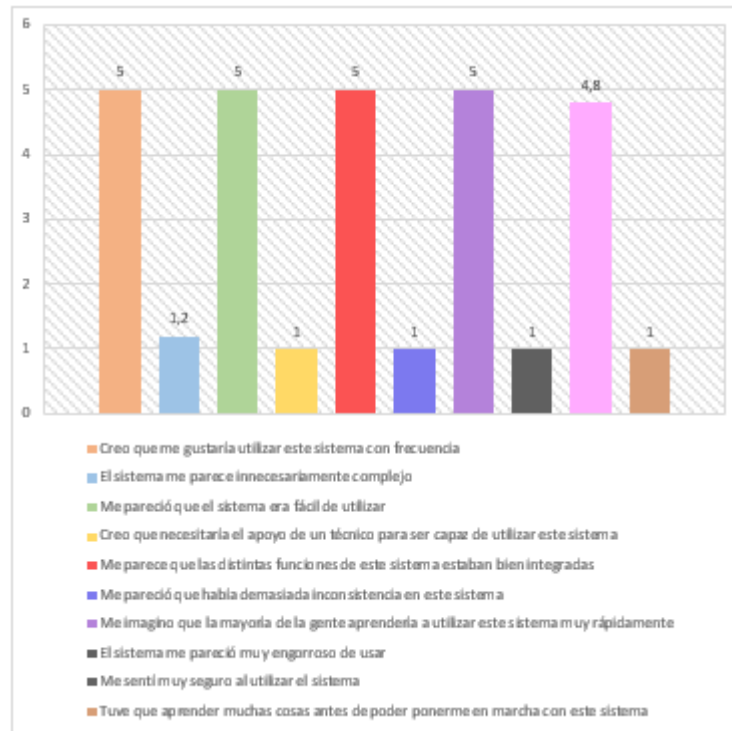


Figura 7.1: Gráfica de resultados de la Escala de Usabilidad de un Sistema

En cuanto a las preguntas sobre Diseño, Funcionalidades individuales y Utilidad real de la aplicación, los resultados fueron muy positivos. El diseño de la aplicación se consideró adecuado y únicamente se sugirió dar colores diferentes a cada una de las pestañas de las funcionalidades, para diferenciarlas mejor. En cuanto a las funcionalidades, la consideración fue, en general, que resultaban fáciles de comprender y utilizar, aunque el tamaño de los pictogramas era algo pequeño y se prefería que pudiera aumentarse. Sobre la utilidad real del proyecto, todas las respuestas apuntaron a que el proyecto era muy útil para el profesorado y expresaron su interés en que siguiera en desarrollo en el futuro. El resultado de la media de cada pregunta en cada apartado se puede consultar en la Figura 7.2.

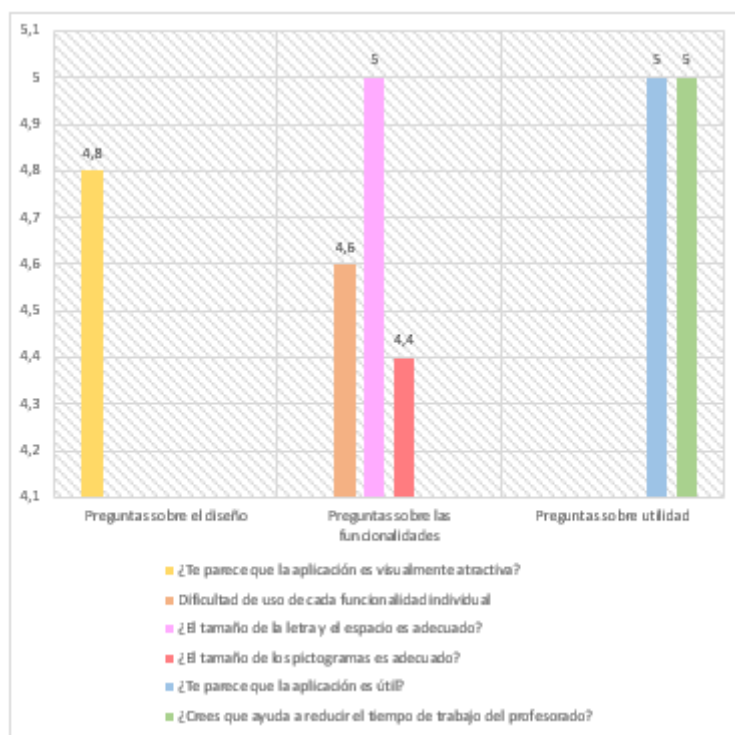


Figura 7.2: Gráfica de resultados de Diseño, Funcionalidades y Utilidad real

Además de reiterar la necesidad de continuar desarrollando los requisitos de la captación inicial, se recopilaban entre las reuniones y en la encuesta nuevas funcionalidades para asignaturas que no se tuvieron en cuenta al principio, y mejoras adicionales para AdaptaMaterialEscolar. Los requisitos fueron los siguientes:

- Añadir una herramienta de pictotraductor como funcionalidad, para traducir de pictogramas a lenguaje natural y viceversa.
- Añadir cuadrícula para ejercicios de matemáticas, para que los alumnos puedan escribir los números en ella.
- Añadir la alternativa de añadir doble pauta, en vez de renglones de una única línea, para determinar el tamaño de la letra del alumno.
- Crear una herramienta de recorte de imágenes para el texto original.
- Añadir encabezado al texto con, al menos, nombre del centro educativo, nombre del alumno y asignatura.
- Añadir ejercicios con espacio para dibujar.

- Añadir ejercicios para ejercicios de cálculo con fórmulas con huecos a rellenar por el alumno.
- Enumerar ejercicios de forma automática.
- Añadir doble pauta en lugar de renglones sencillos, para delimitar el tamaño de la letra del alumno.
- Añadir algún tipo de fuente de letra escolar.

7.5. Conclusión de la evaluación

Con respecto a la evaluación de la aplicación web AdaptaMaterialEscolar, podemos determinar que la acogida por parte de los usuarios finales fue mejor de la esperada y que se ha conseguido desarrollar una herramienta para docentes que tiene una utilidad real que cumple con sus necesidades en el momento de facilitar la adaptación de material escolar. La usabilidad del sistema, con una puntuación de 99 sobre 100, y la estética de la aplicación, con una puntuación de 4,6 sobre 5, requieren poca mejora futura, aunque siempre se podrá perfeccionar si se continua colaborando con nuevos usuarios.

La conclusión que podemos sacar después de haber realizado la evaluación es que el proyecto ha sido un acierto y hay una necesidad real de que se continúe desarrollándolo y mejorándolo, para así cubrir el amplio espectro de peticiones de los docentes y las diversas asignaturas de los centros académicos que podrían beneficiarse del uso de la aplicación.

Aún así, aunque el proyecto generó interés en el profesorado del esperado, el número de evaluaciones que obtuvimos fue muy reducido, por lo que para conseguir una conclusión más significativa se debería contactar con más docentes, para recoger también sus evaluaciones y propuestas de mejora futura para la aplicación.

Capítulo 8

Conclusiones y Trabajo futuro

RESUMEN: En este capítulo se abordarán, en la sección 8.1 las conclusiones de este proyecto, y en la sección 8.2, las posibles modificaciones y mejoras que podrían llevarse a cabo durante los próximos años en la aplicación de AdaptaMaterialEscolar.

8.1. Conclusiones

Actualmente existen carencias en cuanto a la existencia de herramientas de código abierto pensadas para ser utilizadas por el profesorado y facilitar su labor docente durante la adaptación de temario, actividades y exámenes, para aquellos alumnos que necesiten una adaptación curricular a lo largo de su etapa académica. Además, es aún más complicado encontrar aplicaciones que personalicen y sigan un estándar apropiado para los estudiantes con dificultades de aprendizaje.

El objetivo principal de este proyecto era desarrollar una aplicación web que permitiera a los docentes adaptar recursos de las asignaturas de una forma fácil, sencilla y rápida, y crear material personalizado que se ajuste a las necesidades de cada estudiante. Tras ver los problemas que presentaban algunos usuarios finales, fuimos capaces de identificar un problema real y pudimos desarrollar una solución eficiente para esos usuarios. Por un lado integramos un editor de texto de código abierto, *CKEditor*, con el que cumplíamos con la mayoría de los requisitos de formato que nos pedían, tales como tamaño de la letra, color de la letra, subrayados, etc.; y por el otro, implementamos la mayor parte de los ejercicios y herramientas propuestos

que se habían calificado como prioritarios, como el buscador de pictogramas, ejercicios de verdadero/falso, ejercicio de sopa de letras, ejercicio de completar huecos, ejercicio de pregunta para desarrollar y ejercicio de definir un concepto. Gracias también a *CKEditor*, todo el documento adaptado se podía descargar en formato *.pdf*. Así mismo, la aplicación ha seguido un Diseño Centrado en el Usuario (DCU), enseñando a los usuarios finales el diseño del mismo, y dándonos *feedback* para mejorarlo. También participaron en la evaluación los usuarios finales, por lo que pudieron probar la aplicación y así ver la usabilidad y utilidad de la misma.

Otro de los objetivos que tenía este proyecto era aplicar los conocimientos adquiridos durante el Grado de Ingeniería del Software. Consideramos que este objetivo también lo hemos cumplido destacando la aplicación de las siguientes asignaturas:

- **Fundamentos de la Programación y Tecnología de la Programación:** donde adquirimos los conocimientos básicos necesarios para empezar a programar en C++ y Java respectivamente. Estos conocimientos los hemos adaptado al lenguaje de programación utilizado en el del proyecto, Javascript.
- **Estructura de Datos y Algoritmos y Técnicas algorítmicas en ingeniería del software:** donde aprendimos a pensar cómo estructurar y optimizar nuestro código. Esta asignatura nos ha permitido valorar qué estructura de datos es la más adecuada para nuestro proyecto y qué algoritmos son los mejores para tratar con ésta.
- **Ingeniería del Software, Modelado de Software y Gestión de proyectos Software:** donde aprendimos distintos patrones de diseño a la hora de desarrollar código y cómo gestionar y trabajar en equipo de forma correcta en un proyecto software. Esto nos permitió saber elegir una metodología de trabajo adecuada a nuestro proyecto, Kanban, y una arquitectura Redux para la implementación de éste.
- **Aplicaciones web:** donde adquirimos los conocimientos necesarios sobre HTML, CSS y Javascript para desarrollar páginas web. Esta asignatura ha sido fundamental a la hora de diseñar nuestra web y poder desarrollar su comportamiento dinámico.
- **Ética, legislación y profesión:** donde aprendimos, sobre todo, a proteger nuestro código y a cómo usar las licencias y software libre de terceros. Por ello, hemos recurrido a implementar librerías de código abierto en nuestro proyecto, además de decidir que nuestra aplicación sea del mismo tipo.

Durante el desarrollo de la aplicación AdaptaMaterialEscolar hemos adquirido conocimientos nuevos sobre el uso de nuevas tecnologías, como React, muy utilizada actualmente en el mundo laboral. Además, hemos llegado a trabajar con usuarios finales que tenían una necesidad real y hemos conseguido resolverla, incluso llegando a superar sus expectativas. El resultado del proyecto ha tenido una acogida muy positiva por parte de los usuarios finales y se pretende que la aplicación siga en desarrollo durante los próximos años para mejorar su funcionalidad y llegar a un mayor número de docentes, centros educativos o asociaciones que lo pueda necesitar.

8.2. Trabajo futuro

Después de todo el proceso de desarrollo del proyecto, a pesar de que se ha cumplido con la mayoría de los requisitos prioritarios capturados, quedan por completar aún algunas de las tareas que se propusieron inicialmente para considerar que AdaptaMaterialEscolar ha sido completado.

Aunque actualmente nuestra propuesta de Trabajo de Fin de Grado es apta para ser utilizada por los centros educativos, sigue siendo mejorable y hay bastante trabajo futuro a desarrollar, incluso puede que para varios años.

De todos los requisitos iniciales que se obtuvieron, queda pendiente la implementación de los siguientes:

- Adaptaciones de temario

- Generar un resumen a partir de un texto.
- Crear esquemas que faciliten la visualización del temario y/o las actividades, seleccionando contenido de un texto ya redactado o escribiendo desde cero.
- Crear tablas que organicen el temario y/o las actividades, seleccionando contenido de un texto ya redactado o escribiendo desde cero.

- Adaptaciones de actividades y/o exámenes:

- Ejercicios de relacionar contenido mediante flechas.
- Ejercicios de completar los espacios en blanco en tablas y esquemas.

- Adaptaciones de formato:
 - Sustituir una palabra por una imagen.
 - Añadir una leyenda de colores con la categoría de cada tipo de palabra resaltada, para que los alumnos puedan identificar el significado de cada una de las palabras resaltadas en el texto.
 - Añadir leyenda de colores para el tema de cada asignatura, para que los alumnos las distingan con mayor facilidad.
 - Estandarizar formato para títulos e índices del temario, generando de forma automática una plantilla del color de la asignatura.
 - Añadir imágenes buscando una palabra en bases de datos de imágenes libres.

También contábamos con que a partir de la evaluación de la aplicación surgirían nuevas ideas y requisitos para realizar mejoras y, de hecho, ha sido tal la aceptación de AdaptaMaterialEscolar que hemos conseguido más de los esperados. Las peticiones y propuestas de mejora de los docentes que han llegado a evaluar el sistema son:

- Añadir un pictotraductor como funcionalidad.
- Añadir la alternativa de añadir doble pauta, en vez de renglones de una única línea, para determinar el tamaño de la letra del alumno.
- Añadir un tipo de fuente escolar.
- Crear una herramienta de recorte de imágenes para el texto original.
- Añadir encabezado al texto con, al menos, nombre del centro educativo, nombre del alumno y asignatura.
- Añadir ejercicios con espacio para dibujar.
- Añadir ejercicios para ejercicios de cálculo con fórmulas con huecos a rellenar por el alumno.
- Enumerar ejercicios de forma automática.
- Exportar el documento a formato Word para hacer modificaciones.
- Añadir ejercicios de matemáticas con cuadrícula para escribir los números.
- Añadir ejercicios con fórmulas predefinidas con espacio para que el alumno rellene ciertos datos y realice cálculos.

En caso de que se continuara desarrollando el proyecto, sería necesario volver a reevaluar cada requisito, para establecer de nuevo una tabla de prioridades basándose en la importancia y dificultad de cada tarea.

Además, cabe la posibilidad de que incluso se nos invite a colaborar en la formación de nuevo profesorado y alumnos de máster en el IES Maestro Juan de Ávila en los próximos cursos escolares, para enseñarles a utilizar `AdaptaMaterialEscolar`, de manera que puedan incluirlo como herramienta de trabajo de uso diario en los diferentes departamentos del centro.

También sería necesario desarrollar las páginas de Ayuda, con vídeos mostrando cada tipo de funcionalidad a modo de guía, y Contacto de la aplicación web, para ofrecer soporte a cualquier usuario que quiera aprender a utilizarla.

Capítulo 9

Conclusions and Future Work

RESUMEN: This chapter will address, in section 9.1 the conclusions of this project, and in section 9.2, the possible modifications and improvements that could be carried out in the coming years in the application of AdaptaMaterialEscolar.

9.1. Conclusions

Currently, there is a lack of open-source tools designed to be used by teachers to facilitate their teaching work during the adaptation of syllabus, activities, and exams for students who need a curricular adaptation throughout their academic stage. In addition, it is even more complicated to find applications that customize and follow an appropriate standard for students with learning difficulties.

The main objective of this project was to develop a web application that would allow teachers to adapt subject resources in an easy, simple, and fast way, and to create customized material that fits the needs of each student. After seeing the problems presented by some end users, we were able to identify a real problem and were able to develop an efficient solution for those users. On the one hand, we integrated an open-source text editor, *CKEditor*, with which we fulfilled most of the formatting requirements they requested, such as font size, font color, underlines, etc.; and on the other hand, we implemented most of the proposed exercises and tools that had been prioritized, such as the pictogram finder, true/false exercises, word search exercise, fill-in-the-blanks exercise, question-to-develop exercise, and define-a-concept exercise.

Thanks also to *CKEditor*, the whole adapted document could be downloaded in *.pdf* format. Additionally, the application has followed a User-Centered Design (UCD), teaching the end-users the design of the application, and giving us feedback to improve it. End users also participated in the evaluation, so they could test the application and see its usability and usefulness.

Another objective of this project was to apply the knowledge acquired during the Software Engineering Degree. We consider that this objective has also been achieved by highlighting the application of the following subjects:

- **Programming Fundamentals and Programming Technology:** where we acquire the basic knowledge necessary to start programming in C++ and Java respectively. We have adapted this knowledge to the programming language used in the project, Javascript.
- **Data Structure and Algorithms and Algorithmic Techniques in Software Engineering:** where we learned to think about how to structure and optimize our code. This subject has allowed us to assess which data structure is the most suitable for our project and which algorithms are the best to deal with it.
- **Software Engineering, Software Modeling, and Software Project Management:** where we learned different design patterns when developing code and how to manage and work as a team correctly in a software project. This allowed us to choose an appropriate work methodology for our project, Kanban, and a Redux architecture for its implementation.
- **Web applications:** where we acquired the necessary knowledge about HTML, CSS, and Javascript to develop web pages. This subject has been fundamental when designing our web and being able to develop its dynamic behavior.
- **Ethics, legislation, and profession:** where we learned, especially, how to protect our code and how to use third-party licenses and free software. Therefore, we have resorted to implementing open source libraries in our project, in addition to deciding to make our application of the same type.

During the development of the *AdaptaMaterialEscolar* application, we have acquired new knowledge about the use of new technologies, such as React, currently widely used in the working world. In addition, we have worked with end users who had a real need and we have managed to solve it, even exceeding their expectations. The result of the project has been very

positively received by the end-users and we intend to continue developing the application over the coming years to improve its functionality and reach a greater number of teachers, schools, or associations that may need it.

9.2. Future work

After the entire project development process, although most of the priority requirements captured have been met, some of the tasks that were initially proposed still need to be completed to consider AdaptaMaterial-Escolar completed.

Although our Final Degree Project proposal is currently suitable for use by schools, it can still be improved and there is a lot of future work to be done, maybe even for several years.

Of all the initial requirements that were obtained, the following are still pending implementation:

- Agenda adaptations
 - Generate a summary from a text.
 - Create outlines that facilitate the visualization of the agenda and/or activities, selecting content from an already written text or writing from scratch.
 - Create tables that organize the agenda and/or activities, selecting content from an already written text or writing from scratch.
- Adaptations of activities and/or exams:
 - Exercises to relate content using arrows.
 - Exercises to fill in the blanks in tables and diagrams.
- Format adaptations:
 - Replace a word with a picture.
 - Add a color legend with the category of each type of highlighted word, so that students can identify the meaning of each of the highlighted words in the text.
 - Add a color legend for the topic of each subject, so that students can distinguish them more easily.

- Standardize formatting for titles and indexes of the syllabus, automatically generating a template of the subject color.
- Add images by searching for a word in free image databases.

We also expected that from the evaluation of the application new ideas and requirements for improvements would arise and the acceptance of *AdaptaMaterialEscolar* has been such that we have achieved more than expected. The requests and proposals for improvement from teachers who have come to evaluate the system are as follows:

- Add a pictogram translator as a functionality.
- Add the alternative of adding double guideline, instead of single line lines, to determine the student's font size.
- Add a school font.
- Create an image cropping tool for the original text.
- Add a header to the text with at least school name, student's name and subject.
- Add exercises with space to draw.
- Add exercises for calculation exercises with formulas with gaps to be filled in by the student.
- Enumerate exercises automatically.
- Export the document to Word format to make modifications.
- Add math exercises with a grid to write the numbers.
- Add exercises with predefined formulas with space for the student to fill in certain data and to perform calculations.

If the project were to be further developed, it would be necessary to re-evaluate each requirement, to re-establish a table of priorities based on the importance and difficulty of each task.

In addition, we may even be invited to collaborate in the training of new teachers and master students at IES Maestro Juan de Ávila in the next school years, to teach them how to use *AdaptaMaterialEscolar*, so that they can include it as a working tool for daily use in the different departments of the center.

It would also be necessary to develop the Help page, with videos showing each type of functionality as a guide, and the Contact page of the web application, to offer support to any user who wants to learn how to use it.

Capítulo 10

Trabajo Individual

RESUMEN: En este capítulo se habla del trabajo individual que ha realizado cada miembro del equipo en el proyecto.

10.1. Natalia

Una de las tareas que he realizado durante todo el proyecto ha sido la de actuar como punto de comunicación entre nuestro equipo y las profesoras del Aula TEA del IES Maestro Juan de Ávila de Ciudad Real. Contacté con ellas para realizar una entrevista presencial, donde se les pidió colaboración en el proyecto, de forma que pudieran aportar su punto de vista profesional y experiencia trabajando con alumnos con necesidades especiales. Realicé la captación de requisitos inicial del proyecto en base a las necesidades que para las profesoras del Aula TEA aún no estaban cubiertas por ninguna herramienta de trabajo que pudiera facilitar la adaptación de material escolar, aunque tampoco para ningún otro docente de ese centro. Evalué los requisitos que se habían obtenido como resultado de la captación de requisitos de forma individual, según los conocimientos adquiridos durante el grado, y calculé la prioridad final de cada tarea con la puntuación que habían asignado las profesoras y todos los componentes del equipo de desarrollo.

Realicé junto con mis compañeros de equipo un diseño para la interfaz gráfica de la aplicación web para conseguir un prototipo inicial, de forma que vimos las diferentes herramientas que podíamos añadir al trabajo para aportar valor, como por ejemplo, un editor de código abierto de tipo WY-

SIWYG¹.

También realicé un diseño distinto al prototipo inicial del proyecto de forma individual, para que no hubiera influencia entre el equipo y se pudiera debatir luego sobre los cambios que cada uno consideraba importantes, de manera que fuera posible mejorar el diseño inicial añadiendo los cambios que a todos nos habían parecido necesarios y relevantes.

Investigué sobre cómo subir un fichero de texto que estuviera almacenado en una ubicación local del ordenador y sobre el editor de texto CKEditor 5, para poder añadirlo al prototipo individual y comprobar la dificultad inicial de implementar los requisitos necesarios para el proyecto.

Una vez que se consiguió añadir el editor de texto y escogimos la interfaz definitiva, comencé a desarrollar la adaptación de formato del generador de resúmenes y tres actividades: ejercicio de verdadero o falso, de definiciones y de desarrollo.

Cuando tuvimos una versión suficientemente desarrollada de AdaptaMaterialEscolar, me encargué de preparar la evaluación del proyecto y redacté las preguntas que haríamos a los usuarios que probaron el sistema, para poder obtener información real sobre la utilidad de la aplicación y las posibles mejoras que podrían hacerse en un futuro cercano.

Contacté de nuevo con los usuarios finales para realizar una reunión presencial en la que pudiera mostrarles el avance del proyecto, cómo se utilizaba la aplicación y darles la opción de que la evaluaran.

Por último, realicé más demostraciones de AdaptaMaterialEscolar a otros profesores que se interesaron y analicé los resultados obtenidos después de la reunión presencial con las profesoras.

En cuanto a la redacción de la memoria del Trabajo Final de Grado, me encargué del Capítulo 1, con la introducción, motivación y objetivos del proyecto, del Capítulo 6 el apartado 6.1 de Requisitos y el apartado 6.3 de Estructura del código, el Capítulo 7 con la evaluación y el Capítulo 8 con las conclusiones y el trabajo futuro.

¹**WYSIWYG**, acrónimo de *What You See Is What You Get* (en español, “lo que ves es lo que obtienes”). Es una frase aplicada a los procesadores de texto y otros editores con formato, que permiten escribir un documento mostrando directamente el resultado final, frecuentemente el resultado impreso. <https://es.wikipedia.org/wiki/WYSIWYG>

10.2. Jorge

El 4 de diciembre de 2019 el grupo de investigación NIL organizó un *workshop* en la Facultad con docentes de otros centros y asociaciones, en el que se mostraban herramientas tecnológicas inclusivas que se habían desarrollado o estaban en desarrollo. Presenté, en ese *workshop*, AdaptaMaterial-Escolar, donde expliqué el funcionamiento y finalidad de este proyecto, con ejemplos visuales sobre posibles adaptaciones que podrían llegar a realizarse.

Una vez que Natalia consiguió recabar los requisitos junto con las profesoras del Aula TEA, lo primero de todo fue evaluar esos requisitos por ambas partes; las profesoras debían puntuarlos según la importancia que tuviera ese requisito y nosotros lo tuvimos que puntuar por la dificultad de implementación de forma individual y sin influir los unos con los otros. Una vez realizado esto, se hizo la media de dificultad entre los tres integrantes.

A continuación, creamos en conjunto un prototipo inicial a través de la herramienta online *Moqups*, donde surgieron varias vistas de la aplicación, descritas en la sección 6.2.2. Así mismo, diseñé un prototipo individual, al igual que mis compañeros, para poder mejorar ese diseño en conjunto, el cual se puede ver en las figuras 6.7, 6.10, 6.11 y 6.12.

Me encargué de gestionar y reorganizar el tablero *Kanban*, y de que estuviera lo más actualizado posible tras las reuniones con las tutoras.

Así mismo, y como la esencia de este proyecto es realizar adaptaciones, investigamos distintos editores de código abierto de tipo “WYSIWYG”¹, aunque al final uno de mis compañeros localizó uno que cumplía las características que buscábamos, *CKEditor*. Una vez implementado este editor por parte de mi compañero Pablo, cada uno de nosotros se encargó de buscar herramientas y librerías que pudieran facilitar la implementación de los requisitos. En mi caso investigué si había algún paquete NPM (*Node Package Manager*) sobre generación de sopas de letras, y ver si con nuestra tecnología era posible integrarlo.

En la aplicación fui el encargado de implementar el ejercicio de generación de sopas de letras y de completar huecos, y ayudé a Natalia con los ejercicios de pregunta para desarrollar, verdadero o falso y definiciones.

Posteriormente rediseñé la mayoría de la interfaz de usuario de nuestra aplicación, en concreto, la barra de herramientas de los ejercicios (*toolbar*) y las ventanas modales de éstos.

También investigué la API de Google Docs para ver si se podía implementar en el proyecto y así poder subir ficheros en formatos que no sean solo *.pdf* de forma sencilla. Esta funcionalidad no se integró finalmente en la aplicación final.

Con respecto a la redacción de la memoria de este proyecto, me encargué de redactar el Capítulo 4 (Herramientas empleadas), Capítulo 5 (Metodología de desarrollo), del Capítulo 6 (AdaptaMaterialEscolar) los apartados 6.2 (Diseño) y del 6.3 (Implementación) el párrafo de introducción, del 6.3.2 (Funcionalidades) los subapartados 6.3.2.1 (Subir un documento fuente), 6.3.2.2 (Editor), 6.3.2.4 (Sopa de letras), 6.3.2.5 (Definiciones), 6.3.2.6 (Pregunta para desarrollar), 6.3.2.7 (Verdadero o falso) y 6.3.2.8 (Completar huecos) y del Capítulo 8 el apartado 8.2 junto con Natalia.

10.3. Pablo

Aporté mi puntuación de 1 a 3 de dificultad a los requisitos propuestos por las profesoras.

Una vez decidido que el proyecto iba a ser web, había que decidir qué tecnologías FrontEnd usar. Por ello propuse usar React en el proyecto dada su comodidad, sencillez y adecuación para el proyecto. Además, mi previa experiencia usándolo serviría de ayuda para mis compañeros. La propuesta fue aceptada e implementada.

Investigué sobre librerías aptas para poder integrar un editor WYSIWYG¹ en la web. Finalmente propuse CKEditor por su gran cantidad de funciones, soporte y personalización, y fue aceptado por el equipo. Exploré la documentación de CKEditor en profundidad, procurando encontrar qué funcionalidades del framework eran útiles y necesarias para nuestro proyecto.

Posteriormente, realicé un prototipo web usando las tecnologías y librerías web propuestas. Este prototipo estaba realizado con React y CKEditor y mostraba un editor de texto que contenía un buscador de pictogramas integrado. De esta manera podía comprobar la viabilidad del proyecto con las tecnologías propuestas y servir como referencia en la estructura y avance del proyecto. Además, facilitaría el aprendizaje de su uso a mis compañeros. Este prototipo conllevó la investigación por mi parte de la API de ARASAAC, pues fueron sus pictogramas los que se implementaron en el buscador.

Inicialicé un repositorio en GitHub para poder almacenar nuestra plan-

tilla de la memoria, gestionarlo de manera más ordenada y no recurrir a otras herramientas de edición de texto. Este repositorio posteriormente se transfirió al repositorio final del proyecto. Además, propuse la metodología de trabajo en Git, sugiriendo que cada integrante del equipo trabajase en una rama propia y, una vez implementasen una funcionalidad estable, se fusionasen los cambios con la rama master, donde se encuentra la versión final.

Realicé investigaciones sobre patrones de diseño Front-End, hasta finalmente proponer Redux. Tras la aceptación de este por parte de mis compañeros, realicé una amplia investigación sobre el funcionamiento de la arquitectura y realicé la adaptación del proyecto con el patrón, creando un proyecto desde cero, refactorizando y clasificando las clases, creando el Store, los reductores, acciones e importando paquetes necesarios para ello como react-redux, reselect, y redux-thunk. Además, integré las funcionalidades que realizaron mis compañeros en sus respectivos prototipos en este nuevo proyecto, para así tener un proyecto en común en el que trabajar. Posteriormente, expliqué los cambios y el funcionamiento a mis compañeros para que comprendiesen la nueva arquitectura y pudieran continuar con sus nuevas tareas en el nuevo proyecto.

Realicé amplias configuraciones en el proyecto de CKEditor, integrándolo con Redux, permitiendo que el editor internamente reconociese nuestro modelo de datos y poder configurar la exportación a PDF de la manera más rigurosa posible.

Realicé investigaciones sobre cómo poder desplegar la aplicación en el servidor cedido por la universidad hasta conseguirlo. Configuré el servidor instalando node y las librerías necesarias para desplegar una aplicación React. Una vez conseguido, generé un documento explicando las herramientas y comandos necesarios para poder subir la aplicación al servidor y que sirviese de soporte tanto para mis compañeros como para mí.

Por último, contacté por escrito con el equipo de CKEditor para pedir una licencia del editor y solicitar una licencia de la funcionalidad de pago exportar a PDF, las cuales aceptaron a cedérmola gratuitamente dada la naturaleza solidaria y de código abierto de nuestro proyecto.

En cuanto a la redacción de la memoria, me encargué de redactar el Capítulo 3 con el Estado del arte, del Capítulo 6, el apartado 6.2 con la Arquitectura y el subapartado 6.2.2.3 con la funcionalidad del Buscador de pictogramas y traduje a inglés el Capítulo 1 de Introducción y el Capítulo 8 de Conclusiones y Trabajo futuro.

Apéndice A

Tabla de puntuaciones individuales y examen de prueba

RESUMEN: En este anexo se recoge la tabla de puntuaciones individuales para cada requisito de cada miembro del equipo de desarrollo, así como el examen de prueba proporcionado en la evaluación de la aplicación.

Requisito	Puntuación		
	Jorge	Natalia	Pablo
Generar un resumen a partir de un texto	3	2	2
Crear esquemas que faciliten la visualización del temario y/o las actividades, seleccionando contenido de un texto ya redactado o escribiendo desde cero	2	1	2
Crear tablas que organicen el temario y/o las actividades, seleccionando contenido de un texto ya redactado o escribiéndolo desde cero	2	1	2
Ejercicios de relacionar contenido mediante flechas	2	2	2
Ejercicios de sopa de letras	3	3	3
Ejercicios de completar espacios en un texto	2	2	3
Ejercicios de pregunta de desarrollo con un espacio limitado para escribir la respuesta	2	2	3

Tabla A.1 continuada de la página anterior

Ejercicios de verdadero o falso	3	3	3
Ejercicios de completar los espacios en blanco en tablas	2	2	2
Ejercicios de completar los espacios en blanco en esquemas	1	1	1
Añadir enunciados y ejemplos (con o sin pictogramas), donde se explique cómo debe resolverse el ejercicio	3	3	3
Sustituir una palabra por un pictograma	3	2	3
Sustituir una palabra por una imagen	2	2	3
Resaltar una palabra con un color	3	3	3
Añadir una leyenda de colores con la categoría de cada tipo de palabra resaltada, para que los alumnos puedan identificar el significado de cada una de las palabras resaltadas en el texto	3	3	3
Añadir una leyenda de colores para el tema de cada asignatura, para que los alumnos las distingan con mayor facilidad	3	3	3
Estandarizar formato en textos, estableciendo una fuente de texto determinada y un tamaño acorde para cada alumno	3	1	2
Estandarizar formato para títulos e índices del temario, generando de forma automática una plantilla del color de la asignatura	3	1	2
Aumentar el tamaño de la fuente de una parte del texto	3	2	3
Subrayar una parte del texto	3	2	3
Poner en negrita una parte del texto	3	2	3
Añadir imágenes buscando una palabra en bases de datos de imágenes libres	2	3	2
Añadir espacio extra entre líneas para escribir	2	2	2

Tabla A.1: Puntuación individual de cada miembro del equipo de desarrollo para cada requisito

Examen de prueba para la evaluación

Creado a partir de un tema de Ana Alonso Frades y redactado por los desarrolladores de AdaptaMaterialEscolar con la aplicación web.

1. Define los siguientes conceptos:





2. Encuentra 3 palabras escondidas en la sopa de letras.

Z	N	Y	N	B	K	A	U	Y	K	Z	P
B	Y	O	U	H	R	B	A	F	A	B	L
Z	B	G	T	L	Z	R	X	L	I	R	H
R	M	N	R	X	X	U	X	Y	Q	U	R
B	P	D	I	Q	I	D	K	G	N	Q	Q
C	V	F	C	I	Q	D	L	W	L	P	G
W	C	A	I	S	Q	Y	L	A	N	H	T
O	C	W	Ó	L	K	V	J	H	W	R	T
K	E	Z	N	Z	M	F	F	I	A	A	E
R	E	P	R	O	D	U	C	C	I	Ó	N
M	R	E	L	A	C	I	Ó	N	P	V	N

3. Rellena los huecos de las siguientes definiciones con el nombre de cada función vital que has encontrado en el ejercicio anterior:

_____ Los seres vivos se alimentan y crecen.

_____ Se relacionan con otros seres vivos que le rodean.

_____ Se reproducen para que nazcan otros seres vivos de su misma especie.

4. Responde con V si es verdadero o con F si es falso las siguientes frases:

a. Los seres vivos están formados por

☐

b. Las células no se alimentan.

☐

c. Las células se pueden ver con un

☐

d. Las células no se reproducen.

☐

5. Explica las partes que tienen todas las células.

Bibliografía

- BASIL, C. ¿Qué son los sistemas aumentativos y alternativos de comunicación (saac)? https://aulaabierta.arasaac.org/wp-content/uploads/2017/04/que_son_los_saac.pdf, 2017.
- BELLOCH, C. Sistemas de signos. <https://www.uv.es/bellochc/logopedia/NRTLogo8.wiki?7>, 2014.
- CADAH, F. Tipos de adaptaciones curriculares individualizadas (a.c.i.). <https://www.fundacioncadah.org/web/articulo/tipos-de-adaptaciones-curriculares-individualizadas.html>, 2012.
- CADAVID, A. N., MARTÍNEZ, J. D. F. y VÉLEZ, J. M. Revisión de metodologías ágiles para el desarrollo de software. *Prospectiva*, vol. 11, páginas 30–32, Disponible en <http://ojs.uac.edu.co/index.php/prospectiva/article/view/36>.
- DEVIN, F. Sistema de escalas de usabilidad: ¿qué es y para qué sirve? <https://uxpanol.com/teoria/sistema-de-escalas-de-usabilidad-que-es-y-para-que-sirve>, 2017.
- ÓSCAR GARCÍA MUÑOZ. Lectura fácil: Métodos de redacción y evaluación. vol. 1, Disponible en <https://www.plenainclusion.org/sites/default/files/lectura-facil-metodos.pdf>.
- GARZAS, J. Kanban. <https://www.javiergarzas.com/2011/11/kanban.html>, 2011.
- GONZÁLEZ, L. ¿Cómo se organiza el sistema educativo español? <https://www.emagister.com/blog/se-organiza-sistema-educativo-espanol/>, 2020.
- LOGOPEDIA. El uso de los pictogramas. http://www.ponceleon.org/logopedia/index.php?option=com_content&view=article&id=110, 2012.
- MORENO, P., JIMÉNEZ, G. y SÁNCHEZ, A. Diseño de sistemas interactivos. 2019.

RODRÍGUEZ, N. G. *Las Pruebas de Integración como Proceso de la Calidad del Software en el Ámbito de las Telecomunicaciones*. Proyecto de Fin de Carrera, Escuela Politécnica Superior Carlos III (Universidad Carlos III de Madrid), 2015.